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. Lastly, in the Learning Management System, click the “Add Submission” button and paste the URL to your GitHub repository.

**Coding Steps:**

For the final project you will be creating an automated version of the classic card game *WAR.*

1. Create the following classes.
   1. Card
      1. Fields
         1. **value** (contains a value from 2-14 representing cards 2-Ace)
         2. **name** (e.g. Ace of Diamonds, or Two of Hearts)
      2. Methods
         1. Getters and Setters
         2. **describe** (prints out information about a card)
   2. Deck
      1. Fields
         1. **cards** (List of Card)
      2. 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.
   3. Player
      1. Fields
         1. **hand** (List of Card)
         2. **score** (set to 0 in the constructor)
         3. **name**
      2. 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.
   1. 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:**

**package** war;

**public** **class** App {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Player p1 = **new** Player("Ugly people");

Player p2 = **new** Player("Sexy people");

Deck deck = **new** Deck();

deck.shuffle();

**int** deckSize = deck.size();

*dealHands*(p1, p2, deck);

*playGame*(p1, p2, deckSize);

**if** (p1.getScore() > p2.getScore()) {

*winnerIs*(p1, p2);

} **else** **if** (p2.getScore() > p1.getScore()) {

*winnerIs*(p2, p1);

} **else** {

System.***out***.println("Holy crap " + p1.getName() + " and " + p2.getName()

+ " both tied with a score of " + p1.getScore());

}

}

**private** **static** **void** winnerIs(Player winner, Player loser) {

System.***out***.println(winner.getName() + " beat " + loser.getName() + ". " +

winner.getScore() + " to " + loser.getScore() + " points. winning by "

+ (winner.getScore() - loser.getScore()) + " points! ");

}

**private** **static** **void** playGame(Player p1, Player p2, **int** deckSize) {

**for** (**int** turn = 0; turn < deckSize / 2; turn++) {

Card c1 = p1.flip();

Card c2 = p2.flip();

**if** (c1.getValue() > c2.getValue()) {

p1.incrementScore();

} **else** **if** (c2.getValue() > c1.getValue()) {

p2.incrementScore();

}

}

}

**private** **static** **void** dealHands(Player p1, Player p2, Deck deck) {

**for** (**int** i = 0; i < 52; i++) {

**if** (i % 2 == 0) {

p1.draw(deck);

} **else** {

p2.draw(deck);

}

}

}

}

**import java.util.ArrayList;**

**import java.util.List;**

**public class Player {**

**private String name;**

**private int score = 0;**

**private List<Card> hand = new ArrayList<>();**

**public Player(String name) {**

**this.name = name;**

**}**

**public String describe() {**

**return toString();**

**}**

**public Card flip() {**

**if (!hand.isEmpty()) {**

**return hand.remove(0);**

**}**

**throw new IllegalStateException("Hand is empty");**

**}**

**public void draw(Deck deck) {**

**hand.add(deck.draw());**

**}**

**public void incrementScore() {**

**score += 1;**

**}**

**@Override**

**public String toString() {**

**return "Player [name=" + name + ", score=" + score + ", hand=" + hand + "]";**

**}**

**public int size() {**

**return hand.size();**

**}**

**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) {**

**this.score = score;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**}**

**import java.util.ArrayList;**

**import java.util.List;**

**public class Player {**

**private String name;**

**private int score = 0;**

**private List<Card> hand = new ArrayList<>();**

**public Player(String name) {**

**this.name = name;**

**}**

**public String describe() {**

**return toString();**

**}**

**public Card flip() {**

**if (!hand.isEmpty()) {**

**return hand.remove(0);**

**}**

**throw new IllegalStateException("Hand is empty");**

**}**

**public void draw(Deck deck) {**

**hand.add(deck.draw());**

**}**

**public void incrementScore() {**

**score += 1;**

**}**

**@Override**

**public String toString() {**

**return "Player [name=" + name + ", score=" + score + ", hand=" + hand + "]";**

**}**

**public int size() {**

**return hand.size();**

**}**

**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) {**

**this.score = score;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**}**

**package** war;

**public** **class** Card {

**private** String name;

**private** **int** value;

**public** Card(String name, **int** value) {

**this**.name = name;

**this**.value = value;

}

**public** String describe() {

**return** toString();

}

@Override

**public** String toString() {

**return** "Card [name=" + name + ", value=" + value + "]";

}

**public** **int** getValue() {

**return** value;

}

**public** **void** setValue(**int** value) {

**this**.value = value;

}

**public** String getName() {

**return** name;

}

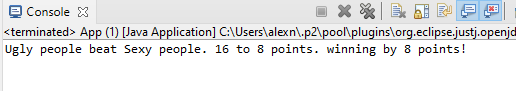
**public** **void** setName(String name) {

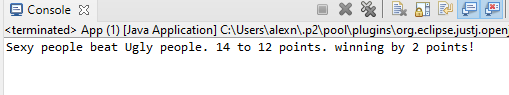
**this**.name = name;

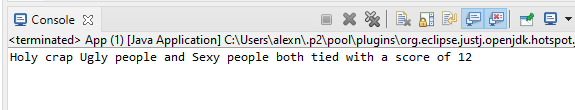
}

}

**Screenshots of Running Application:**

****

****

****

**URL to GitHub Repository:**

**https://github.com/ale2362189/homework\_topic\_6.git**