Intro to Java Week 6 Coding Assignment

Points possible: 100

URL to GitHub Repository: https://github.com/AlexWarr/Week-06-Homework-Final-Java.git

URL to Public Link of your Video: https://youtu.be/5RQfP6BcJGk

———————————————————————————————————————

Instructions:

1. Follow the **Coding Steps** below to complete this assignment.
    - 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.
    - Create a new repository on GitHub for this week's assignment and push your completed code to this dedicated repo.
    - Create a video showcasing your work:
        - In this video: record and present your project verbally while showing the results of the working project.
        - Easy way to Create a video: Start a meeting in Zoom, share your screen, open Eclipse with the code and your Console window, start recording & record yourself describing and running the program showing the results.
        - Your video should be a maximum of 5 minutes.
        - Upload your video with a public link.
        - Easy way to Create a Public Video Link: Upload your video recording to YouTube with a public link.
2. In addition, please include the following in your Coding Assignment Document:
    - The URL for this week's GitHub repository.
    - The URL of the public link of your video.
3. Save the Coding Assignment Document as a .pdf and do the following:
    - Push the .pdf to the GitHub repo for this week.
    - Upload the .pdf to the LMS in your Coding Assignment Submission.

———————————————————————————————————————

Intro to Java Week 6 Coding Assignment

Coding Steps — Java Final Project:

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.
   a) Instantiate a Deck and two Players, call the shuffle method on the deck.
   b) Using a traditional for loop, iterate 52 times calling the Draw method on the other player each iteration using the Deck you instantiated.
   c) Using a traditional for loop, iterate 26 times and call the flip method for each player.
   d) 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.
   e) After the loop, compare the final score from each player.
   f) 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.

3. Tips: Printing out information throughout the game adds value including easier debugging as you progress and a better user experience.
   a) Using the Card describe() method when each card is flipped illustrates the game play.
   b) Printing the winner of each turn adds interest.
   c) Printing the updated score after each turn shows game progression.
   d) At the end of the game: print the final score of each player and the winner's name or "Draw" if the result is a tie.

Code:

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.Scanner;

public class App {
	public static Scanner kb = new Scanner(System.in);
	public static Random rando = new Random();

	public static void main(String[] args) {
		//Welcome to war, this is a simple app to replicate the card game
		String[] names = playerNamer(menu());// determines single or 2 player mode and
names players
		List<Map<String, Integer>> hands = handBuilder(deckBuilder()); //builds the
main deck and then deals it into 2 hands
		Player player1 = new Player(names[0],0, (HashMap<String, Integer>)
hands.get(0)); //creates player 1
		Player player2 = new Player(names[1],0, (HashMap<String, Integer>)
hands.get(1)); // creates player 2

		play(player1,player2); //initiates and carries out gameplay

		System.out.println("Thank you for playing!\n\nWould you like to play again?\n
'y' = play again\n  'n' = exit");
		String again = kb.nextLine();
		//allows players to loop back to beginning for replay
		if (again.contains("y")) {
			App.main(args);
		} else {
			System.out.println("Thank you for playing!\nGood Bye!");
		}


	}

// Methods
	public static boolean menu() {
		//selects number of players
		System.out.println("Welcome to War!\n");
		System.out.println("Please select a mode of play: \n  Enter '1' for Single Player\n
Enter '2' for 2 Players\n  Enter '0' to quit");
		String initchoice = kb.nextLine();
		int choice = 1;
		try {// attempts to catch improper responses
			choice = Integer.parseInt(initchoice);
		} catch (NumberFormatException e) {
			System.out.println("Please choose a valid option");
			System.out.println("  Enter '1' for Single Player\n  Enter '2' for 2 Players\n
Enter '0' to quit");
			choice = Integer.parseInt(kb.nextLine());
		}
		while (choice > 3 || choice < 0) {
			System.out.println("Please choose a valid option");
			System.out.println("  Enter '1' for Single Player\n  Enter '2' for 2 Players\n
Enter '0' to quit");
			choice = Integer.parseInt(kb.nextLine());
		}
		boolean single = true;
		if (choice == 0) {// exits game
			System.out.println("Good Bye!");
			System.exit(choice);
		}
		switch (choice) {
		case 1:
			single = true;
			break;
		case 2:
			single = false;
			break;
			}
		return single;
	}

	public static String[] playerNamer(boolean menu) {
		//names players
		System.out.println("Please enter name for player 1: ");
		String P1 = kb.nextLine();
		String P2 = "";
		if (menu) {
			P2 = "PC";
		} else {
			System.out.println("Please enter name for player 2: ");
			P2 = kb.nextLine();
		}
		String[] names = new String[] {P1,P2};
		return names;


	}

	public static Deck deckBuilder() {
		//creates a HashMap of all faces and values of a regular playing deck minus the
jokers
		String[] royals = new String[]{"Jack","Queen","King","Ace"};
		String[] cases = new String[] {"Hearts","Clubs","Spades","Diamonds"};
		HashMap<String,Integer> stack = new HashMap<String,Integer>();
```

```
        for(int i = 2; i < 15; i++) {
            for (int j = 0; j < 4; j++) {
                String temp = "";
                if (i < 11) {
                    temp = i+ " of " + cases[j];
                    stack.put(temp, i);
                    }
                if (i > 10) {
                    temp = royals[i-11]+ " of " +cases[j];
                    stack.put(temp, i);
                    }
                }

            }
        //creates the main Deck from which the hands will be drawn
        Deck mainDeck = new Deck();
        mainDeck.setCards(stack);
        return mainDeck;
    }


    public static List<Map<String, Integer>> handBuilder(Deck mainDeck) {
        //split mainDeck evenly into two random hands
        HashMap<String,Integer> mDeck = mainDeck.getCards();
        HashMap<String,Integer> hand1 = new HashMap<String,Integer>();
        HashMap<String,Integer> hand2 = new HashMap<String,Integer>();
        boolean error = true; // emplaced to prevent errors with the random number
        Object[] arry = mDeck.keySet().toArray();// iterable array of keys
        int counter = arry.length;
        int shuffler =0;
        while (counter > 0) {
            while (error == true) {
                //pulls a random card from the main deck and places it in hand1
                shuffler = rando.nextInt(counter);
                if (shuffler < mDeck.size()) {
                    hand1.put((String) arry[shuffler],
mDeck.get(arry[shuffler]));
                    mDeck.remove(arry[shuffler]);
                    arry = mDeck.keySet().toArray();
                    counter --;
                    error = false;
                } else{
```

```
							error = true;//if random number fails as index, will simply
retry until successfull
						}
					}
					while (error == false) {
						shuffler = rando.nextInt(counter);
						if (shuffler < mDeck.size()) {
							// pulls a random card from the main deck and places it in
hand2
							hand2.put((String) arry[shuffler],
mDeck.get(arry[shuffler]));
							mDeck.remove(arry[shuffler]);
							arry = mDeck.keySet().toArray();
							counter --;
							error = true;
						} else {
							error = false;}
					}
				}
				List<Map<String, Integer>> hands = new ArrayList<Map<String, Integer>>();
				//allows this method to produce two separate hands with no duplicates and no
missing cards
				hands.add(hand1);
				hands.add(hand2);
				return hands;

			}

			private static void play(Player player1, Player player2) {
				System.out.println(player1.getName() + " vs " + player2.getName());
				System.out.println("\npress enter to begin\n\n or enter 'x' to quit");
				String choice = kb.nextLine();
				if (choice.contains("x")){
					System.out.println("Good Bye!");
					System.exit(0);
				}
				draw(player1,player2);
				win(player1,player2);

			}

			private static void draw(Player player1, Player player2) {
				Card p1 = new Card();
```

```
		Card p2 = new Card();

		int pile = player1.getHand().size();
		HashMap<String, Integer> hand1 = player1.getHand();
		HashMap<String, Integer> hand2 = player2.getHand();

		Object[] arry1 = hand1.keySet().toArray();
		Object[] arry2 = hand2.keySet().toArray();

		while (pile > 0) {
			//shuffles deck to ensure random outcome
			int shuffler = rando.nextInt(pile);
			//recreates P1 card for specific battle
			p1.setFace(arry1[shuffler].toString());
			p1.setValue((int) hand1.get(arry1[shuffler].toString()));

			//recreates P2 card for specific battle
			p2.setFace(arry2[shuffler].toString());
			p2.setValue((int) hand2.get(arry2[shuffler].toString()));


			int d1 = p1.getValue();
			int d2 = p2.getValue();
			//displays battle outcome
			System.out.println(player1.getName() + " draws: " + p1.getFace());
			System.out.println(player2.getName() + " draws: " + p2.getFace());

			//determines victory for battle
			if (d1 == d2){

				//System.out.println(d1 + "==" + d2); //tested to check if the card
builder was accurately valuing cards
				System.out.println("DRAW!");
			} else if (d1 > d2) {
				System.out.println(player1.getName() + " beats " +
player2.getName() + "!");
				player1.setScore(player1.getScore()+1);
			} else if (d1 < d2) {
				System.out.println(player2.getName() + " beats " +
player1.getName() + "!");
				player2.setScore(player2.getScore()+1);
			} else {
				System.out.println("there is an error");
```

```
					break;
				}
				//discards used cards
				hand1.remove(p1.getFace());
				hand2.remove(p2.getFace());
				arry1 = hand1.keySet().toArray();
				arry2 = hand2.keySet().toArray();
				//declares running status
				System.out.println("The score is now: " + player1.getScore() + " to " + 
player2.getScore());
				pile = pile-1;
			}

	}

	private static void win(Player player1, Player player2) {
		//determines victor for game
		System.out.println("The final scores are: \n   " +player1.getName() + ": " + 
player1.getScore() +"\n   " + player2.getName() + ": " + player2.getScore());
		if ( player1.getScore() == player2.getScore()) {
			System.out.println("This round was a draw. Better luck next time.");
			Player.describe(player1.getName(), player1.getScore(), 
player1.getHand());
			Player.describe(player2.getName(), player2.getScore(), 
player2.getHand());
		} else if ( player1.getScore() > player2.getScore()) {
			System.out.println(player1.getName() + " wins!\n");
			Player.describe(player1.getName(), player1.getScore(), 
player1.getHand());
		} else if (player1.getScore() < player2.getScore()) {
			System.out.println(player2.getName() + " wins!\n");
			Player.describe(player2.getName(), player2.getScore(), 
player2.getHand());
		}
	}

	public static void printDeck(HashMap<String,Integer> deck) {
		//use for printing deck to test proper shuffling
		for (String key : deck.keySet()) {
			System.out.println(key + " = " + deck.get(key));
		}
	}
}
```

```
import java.util.HashMap;

public class Player {
	private String name;
	private int score;
	private HashMap<String,Integer> hand;

	public Player() {
		name = "";
		score = 0;
		hand = null;

	}

	public Player(String name, int score, HashMap<String,Integer> hand) {
		this.name = name;
		this.score = score;
		this.hand = hand;
	}

	public String playerDisplay() {
		return "Name: " + getName() + " , Score: " + getScore();
	}

	public void win(int point) {
		setScore(getScore() + point);
	}

	public String getName() {
		return name;
	}

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

	public int getScore() {
		return score;
	}

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

	public HashMap<String,Integer> getHand() {
		return hand;
	}

	public void sethand(HashMap<String,Integer> hand) {
		this.hand = hand;
	}

	public static void describe(String name, int score, HashMap<String,Integer> hand) {
		System.out.println(name + " currently has " + score + " points");
		System.out.println(name + " has the following cards in their hand:");
		Card temp = new Card();
		for (String key : hand.keySet()) {
			temp.setFace(key);
			temp.setValue(hand.get(key));
			Card.describe(temp.getFace(),temp.getValue());
		}
	}


}


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

	public Card() {
		face = "";
		value = 0;
	}

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

	public String getFace() {
		return face;
	}
	public void setFace(String face) {
		this.face = face;
	}

	public int getValue() {
		return value;
	}

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

	public static void describe(String face, int value) {
		System.out.println(face + " has a value of " + value);
	}

}

import java.util.HashMap;

public class Deck {
	private HashMap<String,Integer> cards;

	public Deck() {
		cards = null;

	}

	public Deck(HashMap<String,Integer> cards) {
		this.cards = cards;
	}

	public HashMap<String,Integer> getCards() {
		return cards;
	}

	public void setCards(HashMap<String,Integer> cards) {
		this.cards = cards;
	}

}
```