Intro to JavaScript Week 6 Coding Assignment

Points possible: 100

URL to GitHub Repository: https://github.com/anagalacticRuby/War_Card_Game

URL to Your Coding Assignment Video: https://youtu.be/Gtt-jOtc8g4

Instructions: In Visual Studio Code, 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 JavaScript 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*. You do not need to accept any user input, when you run your code, the entire game should play out instantly without any user input.

There are many versions of the game *WAR*, but in this version there are only 2 players and you don't need to do anything special when there is a tie on a round.

Think about how you would build this project and write your plan down. Consider classes such as Card, Deck, and Player and what fields and methods they might each have. You can implement the game however you'd like (i.e. printing to the console, using alert, or some other way). The completed project should, when run, do the following:

- Deal 26 Cards to two Players from a Deck.
- Iterate through the turns where each Player plays a Card
- The Player who played the higher card is awarded a point
    - Ties result in zero points for both Players
- After all cards have been played, display the score and declare the winner.

Write a Unit Test using Mocha and Chai for at least one of the functions you write.

PROMINEO TECH

Video Steps:

Create a video, up to five minutes max, showing and explaining how your project works with an emphasis on the portions you contributed. This video should be done using screen share and voice over. This can easily be done using Zoom, although you don't have to use Zoom, it's just what we recommend. You can create a new meeting, start screen sharing, and start recording. This will create a video recording on your computer. This should then be uploaded to a publicly accessible site, such as YouTube, Dropbox, or Google Drive. MAKE SURE THE LINK YOU SHARE IS PUBLIC or UNLISTED. If it is not accessible by your grader, your project will be graded based on what they can access. The link should be pasted in the submission text box after the GitHub repo link. REQUIRED: PUBLIC link to video, and GitHub repo link with everything listed above!

Screenshots of Code:

```
//The Card class defines the essential components that will make up Card for the game
of war.
class Card {
  constructor() {
    this.cardSuit = ["Hearts", "Diamonds", "Spades", "Clubs"];
    this.cardValue = [];
    //Cards have their own suit, and hold a value
    //Jacks, Queens, Kings, and Aces will be stored as their numerical equivalent so
    it's easier to manage.
    //Meaning the values range from 1 to 13
    //There cannot be more than 4 cards with the same value. A normal deck of 52 cards
    features 4 suits, and 52 / 4 = 13.
  }
  assignValue() {
    //assignValue will populate an array that will hold 52 values, to represent 52
    cards in a deck.
    //Once this is completed, the array can then be used for the Deck object.
    for (let i = 0; i < this.cardSuit.length; i++) {
      //console.log(`The current loop iteration is: ${i}`); //Prints current loop
      iteration to console
      console.log(`Assigning values for ${this.cardSuit[i]} cards:`);
      for (let j = 0; j < 13; j++) {
        //Should run 13 times, because there are card values 1 through 13
        //console.log(`The current loop iteration is ${i} of ${j}`);
        console.log(`A ${[j+1]} of ${this.cardSuit[i]} has been created.`);
        this.cardValue.push(j + 1);
      }
    }
    console.log(this.cardValue);
    //This prints to console to make sure 52 cards have been created
  }
}
```

```
//The Deck class houses an array of Card objects, and has methods to manipulate those cards & populate a Deck with that array.
class Deck {
  constructor() {
    this.cardStack = [];
    //Decks will have an array property that will store an array of cards currently in them.
    //It makes sense to think of Deck objects as Stacks, because like the programming stack, cards are drawn off the top.
  }

  //The code for the shuffleDeck function was obtained from https://stackoverflow.com/questions/2450954/how-to-randomize-shuffle-a-javascript-array
  shuffleDeck(deckArray) {
    let currentIndex = deckArray.length,
      randomIndex;

    // While there remain elements to shuffle.
    while (currentIndex != 0) {
      // Pick a remaining element.
      randomIndex = Math.floor(Math.random() * currentIndex);
      currentIndex--;

      // And swap it with the current element.
      [deckArray[currentIndex], deckArray[randomIndex]] = [
        deckArray[randomIndex],
        deckArray[currentIndex],
      ];
    }
    return deckArray;
  }

  fillDeck(cardsArray) {
```

```
  fillDeck(cardsArray) {
    for (let i = 0; i < cardsArray.length; i++) {
      this.cardStack.push(cardsArray[i]);
    }
    //An array of Card objects are passed in as a parameter for this method
    //Then that array of Card objects are pushed into the cardStack array property of the Deck class.
    //Once the Deck is fully populated, it can then be shuffled or manipulated
  }
```

PROMINEO TECH

```
//The player class consists of properties that players will have, and actions they can
perform.
class Player {
  constructor(playerName) {
    this.playerDeck = [];
    this.playerScore = 0;
    this.playerTopCard = null;
    this.playerName = playerName;
    //Players start with an empty deck
    //And a score of 0, both the deck and score will be updated later
    //This game of war will only feature two players, but a player's deck should be
    equal to 1 / x of a 52 card deck, where x = the number of players.
    //So in a game with only 2 players, each player will get 26 cards total.
    //playerTopCard is set to null for now, it will later store the value of a card
    drawn from the top of the player's deck.
  }
```

```
dealDeck(deckArray) {
  for (let cardsDealt = 0; cardsDealt != 26; cardsDealt++) {
    //I chose to use a for loop here so it is easier to visualize what is going on
    in this function.
    //The for loop will run until it has dealt 26 cards, which is half of a 52 card
    deck.
    //It should be noted that rather than run the loop based on cards remaining in
    the deck, the foor loop runs until it reaches a count of 26.
    //This will make sure that no more than 26 cards are dealt to a player.
    //However I am aware this function will cause problems if the array passed into
    it does not have 26 or more elements to remove
    //But that is intentional because this program is designed with a 52 card deck
    in mind.
    let topCard = deckArray.pop();
    //Take the 'top' card from the Deck and store it in a variable.
    //By using pop(), the deckArray will update and remove the last element,
    returning that removed element and storing it in topCard.
    this.playerDeck.push(topCard);
    //After topCard is set to the value removed by calling pop() on the passed in
    deckArray
    //push the topCard into the player's Deck. When this loop finishes, the
    playerDeck array should store 26 values, correlating to their 26 cards.
  }
}
drawCard() {
  this.playerTopCard = this.playerDeck.pop();
  //"Draw" the top card of the player's deck by calling pop() on playerDeck
  //The top card "Drawn" is then stored in playerTopCard
  return this.playerTopCard;
  //playerTopCard is returned by drawCard so when it is called by other methods it
  can be used for comparison
```

```
class Game {
  //Rules of the game:
  //There are 2 players
  //Each player is dealt 26 Cards from a Deck
  //Turns are iterated through, where the top card of each player's deck is drawn
  //The higher card value of the two wins, and that player is given a point.
  //Draws result in no points being given to either player for simplicity sake
  //The game ends when there are no more cards that can be drawn (Should be 26 turns
  since there are only 2 players with 26 Cards each)
  //A winner is declared, whatever Player has more points is the winner
}
```

PROMINEO TECH

```
function takeTurn(playerOne, playerTwo) {
  //Check to make sure both players have cards left in their deck before the turn
  starts
  let playerOneDraws = playerOne.drawCard();
  let playerTwoDraws = playerTwo.drawCard();
  //Draw the top card of both player's decks, store the returned values in variables
  to use in takeTurn()
  console.log(
    `Player one draws: ${playerOneDraws} \n Player two draws: ${playerTwoDraws}`
  );
  //Print what both players drew to console
  if (playerOneDraws === playerTwoDraws) {
    console.log("There is a draw. Neither player gets a point.");
    //If the cards drawn are equal in value, no player gets a point.
    //Print outcome of the draws to console
  } else if (playerOneDraws > playerTwoDraws) {
    console.log("Player one gets a point!");
    playerOne.playerScore++;
    //If player 1's drawn card has a value greater than player 2's drawn card, updat
    player 1's score
    console.log(
      `Player one's current score: ${playerOne.playerScore} \n Player two's current
      score: ${playerTwo.playerScore}`
    );
    //Then print the current scores to console
  } else {
    console.log("Player two gets a point!");
    playerTwo.playerScore++;
    //Otherwise if player 2's drawn card has a value greater than player 1's drawn
    card, update player 2's score
    console.log(
      `Player one's current score: ${playerOne.playerScore} \n Player two's current
      score: ${playerTwo.playerScore}`
    );
```

PROMINEO TECH

```
//Here is the code that will create the cards, populate a deck, deal cards to players,
and then start taking turns.
firstCards = new Card();
firstCards.assignValue();
//Step 1: Create Cards
//Step 2: Assign values to those cards

//console.log(firstCards); //Used to debug, making sure firstCards works properly

gameDeck = new Deck();
gameDeck.fillDeck(firstCards.cardValue);
//console.log(gameDeck); //Used to debug, making sure gameDeck was populated with Cards
gameDeck.shuffleDeck(gameDeck.cardStack);

//console.log(gameDeck); //Used for debug, to print the contents of gameDeck

//Step 3: Create a Deck
//Step 4: Populate Deck with the previously created and defined Cards from step 2
//Step 5: Randomize order of the Deck

playerOne = new Player("Player One");
playerTwo = new Player("Player Two");

playerOne.dealDeck(gameDeck.cardStack);
playerTwo.dealDeck(gameDeck.cardStack);
//Step 6: Create 2 Players, make sure to give them their own name
//Step 7: Deal cards to both players, using the shuffled Deck from step 5
```

```
while (
  (playerOne.playerDeck.length !== 0) &
  (playerTwo.playerDeck.length !== 0)
) {
  takeTurn(playerOne, playerTwo);
  //Step 8: Until both players have an empty deck, take turns drawing cards and giving players
  points.
}
console.log("There are no more cards to draw! Time to declare a winner.");
if (playerOne.playerScore === playerTwo.playerScore) {
  console.log(
    `There is a Tie! How shocking! Both players had a score of ${playerOne.playerScore}`
  );
  //Print a custom message for tied games
  if(playerOne.playerScore === 13 && playerTwo.playerScore === 13){
    console.log(`Wow! A perfect game! That's incredibly rare.`);
  }
  //In the rare event of a perfect tie, a unique message is printed along with the previous tie
  game message.
} else {
  let winningPlayer =
    playerOne.playerScore > playerTwo.playerScore ? playerOne : playerTwo;
  //Step 9: After there are no more cards to draw (or turns to take), assign a winner
  //Here, winningPlayer is assigned using a ternary operator
  //Allowing a winner to be assigned without writing lots of code
  //The result of the inequality will determine the result stored in winningPlayer
  console.log(`The final scores of both players are: \n
    ${playerOne.playerName}'s score: ${playerOne.playerScore} \n
    ${playerTwo.playerName}'s score: ${playerTwo.playerScore} \n
    The winner is ${winningPlayer.playerName}, with their score of ${winningPlayer.playerScore}
    !`);
  //Step 10: Print the final scores for both players, then print the name and score of the
  winning player.
```

Screenshots of Running Application:

(This is what it looks like as the code creates Cards and assigns them values. This happens 4 times in total, for each card suit.)

```
Assigning values for Hearts cards:
A 1 of Hearts has been created.
A 2 of Hearts has been created.
A 3 of Hearts has been created.
A 4 of Hearts has been created.
A 5 of Hearts has been created.
A 6 of Hearts has been created.
A 7 of Hearts has been created.
A 8 of Hearts has been created.
A 9 of Hearts has been created.
A 10 of Hearts has been created.
A 11 of Hearts has been created.
A 12 of Hearts has been created.
A 13 of Hearts has been created.
```

(This is a snippet showing two players drawing a card. An instance of a draw happening also appears here)

```
Player one draws: 4
 Player two draws: 6
Player two gets a point!
Player one's current score: 2
 Player two's current score: 1
Player one draws: 7
 Player two draws: 7
There is a draw. Neither player gets a point.
```

(Below is what happens when both players run out of cards, and a winner is declared.)

```
Player one's current score: 15
 Player two's current score: 10
There are no more cards to draw! Time to declare a winner.
The final scores of both players are:

    Player One's score: 15

    Player Two's score: 10

    The winner is Player One, with their score of 15!
```

(A really rare perfect tie, showing the custom tie message)

```
There are no more cards to draw! Time to declare a winner.
There is a Tie! How shocking! Both players had a score of 13
Wow! A perfect game! That's incredibly rare.
```

Screenshots of Test Code:

```
var expect = chai.expect;

describe("My Functions", function () {
  describe("#assignValue", function () {
    it("Should make sure that 52 card values are made.", function () {
      let testCards = new Card();
      testCards.assignValue();
      expect(testCards.cardValue.length).to.equal(52);
    });
  });
```

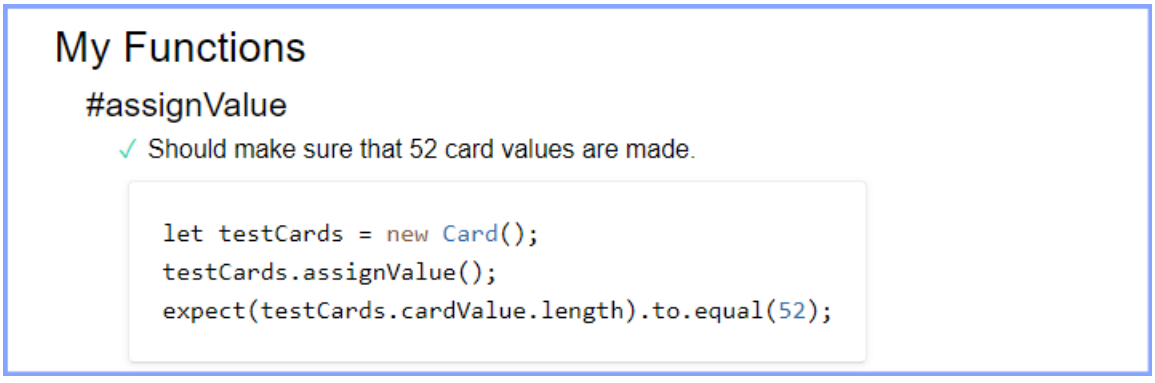
```
describe("#takeTurn", function(){
  it("Should draw the top two cards of 2 player's decks.",function(){
      console.log("Testing the takeTurn function...")

      let playerOne = new Player();
      let playerTwo = new Player();
      //Two player objects are required to use takeTurn()
      playerOne.playerDeck.push(5);
      playerTwo.playerDeck.push(6);
      //Those player objects also need cards in their deck to be drawn

      takeTurn(playerOne,playerTwo);
      //Call the takeTurn method for testing

      expect(playerTwo.playerDeck.length).to.equal(0);
      expect(playerOne.playerDeck.length).to.equal(0);
      //If this method works, we expect both players decks to be empty
  })
  it("Should add one point to a player's score", function(){
      console.log("The takeTurn function should add one point to a player's score.")
      let playerOne = new Player();
      let playerTwo = new Player();
      //New player objects need to be created in order to use takeTurn
      playerOne.playerDeck.push(4);
      playerTwo.playerDeck.push(13);
      //For simplicity sake, we'll make both players have 1 card each
      takeTurn(playerOne,playerTwo);
      //call takeTurn with the recently created & populated players
      expect(playerTwo.playerScore).to.equal(1);
      //Player 2 should have 1 point, because 13 > 4
```

Screenshots of Test Results:



PROMINEO TECH

#takeTurn

✓ Should draw the top two cards of 2 player's decks.

```
console.log("Testing the takeTurn function...")

let playerOne = new Player();
let playerTwo = new Player();
//Two player objects are required to use takeTurn()
playerOne.playerDeck.push(5);
playerTwo.playerDeck.push(6);
//Those player objects also need cards in their deck to be drawn
takeTurn(playerOne,playerTwo);
//Call the takeTurn method for testing
expect(playerTwo.playerDeck.length).to.equal(0);
expect(playerOne.playerDeck.length).to.equal(0);
//If this method works, we expect both players decks to be empty
```

✓ Should add one point to a player's score

```
console.log("The takeTurn function should add one point to a player's score.")
let playerOne = new Player();
let playerTwo = new Player();
//New player objects need to be created in order to use takeTurn
playerOne.playerDeck.push(4);
playerTwo.playerDeck.push(13);
//For simplicity sake, we'll make both players have 1 card each
takeTurn(playerOne,playerTwo);
//call takeTurn with the recently created & populated players
expect(playerTwo.playerScore).to.equal(1);
//Player 2 should have 1 point, because 13 > 4
```