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*.

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 ran, 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 either Player
- After all cards have been played, display the score.

Write Unit Tests using Mocha and Chai for each of the functions you write.

PROMINEO TECH

Screenshots of Code:

deck.js > playRoundResults

```
const SUITS = ["club", "diamond", "heart", "spade"]
const VALUES = ['A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K']
const cardValueMap = {
    '2': 2,
    '3': 3,
    '4': 4,
    '5': 5,
    '6': 6,
    '7': 7,
    '8': 8,
    '9': 9,
    '10': 10,
    'J': 11,
    'Q': 12,
    'K': 13,
    'A': 14
}

class Card {

    constructor(suit, value) {
        this.suit = suit;
        this.value = value;
    }
}

class Player {

    constructor(name) {
        this.name = name;
        this.playerDeck = [];
        this.playerScore = 0;
    }

    addNewDeck(deck) {
        this.playerDeck = deck;
    }
}
```

PROMINEO TECH

deck.js > playRoundResults

```
34
35    addNewDeck(deck) {
36      this.playerDeck = deck;
37    }
38  }
39
40  class Deck {
41
42    constructor(cards = freshDeck()) {
43      this.cards = cards;
44    }
45
46    //get allows us to assign numberofcards = this.cards.length so it doesnt have to be repeated as often
47    get numberOfCards() {
48      return this.cards.length;
49    }
50    //method shuffle to shuffle cards up in a random order basically looping and swapping thru the cards
51    shuffle() {
52      //To achieve the better sorting/ shuffling results we will create a for loop
53      //using this.numberOfCards will be a better way to see it in plain english
54      //i > 0; because we do not need to flip the final card.
55      //in plain terms we are going from the back of the cards to the front of the cards i-- for removing cards till the game ends
56      for (let i = this.numberOfCards - 1; i > 0; i--) {
57        //in order to get a random card we will random index that is eariler in the deck of cards aka math.random, then we want to multiply it by
58        //i which is the current index + 1. This will give us a placement inside of our deck that is somewhere else
59        //to ensure it is an interger we will use math.floor
60        //changed to this.numberOfCards to go thru every card better
61        const newIndex = Math.floor(Math.random() * (this.numberOfCards));
62        //now we want to flip the values at the new index with the current index,so we need the oldValue = basically the value currently at our newindex
63        const oldValue = this.cards[newIndex];
64        //now we need to take the card that is at our i index and put it where our new index is.
65        this.cards[newIndex] = this.cards[i];
66        this.cards[i] = oldValue;
67      }
68    }
69  }
70
```

deck.js > playRoundResults

```
69  }
70
71
72  function freshDeck() {
73    //using a flat map makes a nice and neat array rather than just map that will give you 4 seperate arrays
74    return SUITS.flatMap(suit => {
75      return VALUES.map(value => {
76        return new Card(suit, value);
77      });
78    });
79  }
80
81
82  function setupGame(player1, player2) {
83    //create a deck
84    const deck = new Deck();
85    //shuffles cards
86    deck.shuffle();
87
88    //splits deck of cards and gives a variable to use for splitting the deck
89    const middleOfDeck = (deck.numberOfCards / 2);
90    //this creates a deck thats already preshuffled (more information on the array slice method)
91    //https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice
92    player1.addNewDeck(deck.cards.slice(0, middleOfDeck));
93    //since we only have the last 26 cards left we have to start from middleOfDeck-end aka deck.numberOfCards
94    player2.addNewDeck(deck.cards.slice(middleOfDeck, deck.numberOfCards));
95    //checking the shuffled decks of the players
96
97  }
98
99
100 function roundOutput(player1, player2, roundNum) {
101   console.log(`${player1.name} plays: ${player1.playerDeck[roundNum].value} of ${player1.playerDeck[roundNum].suit}
102   `);
103   console.log(`${player2.name} plays: ${player2.playerDeck[roundNum].value} of ${player2.playerDeck[roundNum].suit}
104   `);
105 }
```

PROMINEO TECH

deck.js > playRoundResults

```
//since we only have the last 26 cards left we have to start from middleOfDeck-end aka deck.numberOfCards
player2.addNewDeck(deck.cards.slice(middleOfDeck, deck.numberOfCards));
//checking the shuffled decks of the players

}


function roundOutput(player1, player2, roundNum) {
  console.log(`${player1.name} plays: ${player1.playerDeck[roundNum].value} of ${player1.playerDeck[roundNum].suit}
  `);
  console.log(`${player2.name} plays: ${player2.playerDeck[roundNum].value} of ${player2.playerDeck[roundNum].suit}
  `);
}


function playRoundResults(player1, player2) {
  //changed to player1.playerDeck.length to create more tests
  for (let i = 0; i < player1.playerDeck.length; i++) {
    roundOutput(player1, player2, i);
    if (cardValueMap[player1.playerDeck[i].value] > cardValueMap[player2.playerDeck[i].value]) {
      player1.playerScore += 1;
      console.log(`${player1.name} has won this round`);
    } else if (cardValueMap[player1.playerDeck[i].value] < cardValueMap[player2.playerDeck[i].value]) {
      player2.playerScore += 1;
      console.log(`${player2.name} has won this round`);
    } else {
      console.log("This is a tie, no points rewarded");
    }
  }
}


function finalTally(player1, player2) {
  if (player1.playerScore > player2.playerScore) {
    console.log(`${player1.name} has won this round with a final score of: ${player1.playerScore}`);
  } else if (player1.playerScore < player2.playerScore) {
    console.log(`${player2.name} has won this round with a final score of: ${player2.playerScore}`);
  } else {
```

PROMINEO TECH

```
function playRoundResults(player1, player2) {
  //changed to player1.playerDeck.length to create more tests
  for (let i = 0; i < player1.playerDeck.length; i++) {
      roundOutput(player1, player2, i);
    if (cardValueMap[player1.playerDeck[i].value] > cardValueMap[player2.playerDeck[i].value]) {
      player1.playerScore += 1;
      console.log(`${player1.name} has won this round`);
    } else if (cardValueMap[player1.playerDeck[i].value] < cardValueMap[player2.playerDeck[i].value]) {
      player2.playerScore += 1;
      console.log(`${player2.name} has won this round`);
    } else {
      console.log("This is a tie, no points rewarded");
    }
  }
}


function finalTally(player1, player2) {
  if (player1.playerScore > player2.playerScore) {
    console.log(`${player1.name} has won this round with a final score of: ${player1.playerScore}`);
  } else if (player1.playerScore < player2.playerScore) {
    console.log(`${player2.name} has won this round with a final score of: ${player2.playerScore}`);
  } else {
    console.log(`${player1.name} and ${player2.name} have tied!`);
  }
}

let Molly = new Player("Molly");
let Kelly = new Player("Kelly");

//Needed to pass in Molly and Kelly once they were called outside of setupGame function to properly run
setupGame(Molly, Kelly);

playRoundResults(Molly, Kelly);

finalTally(Molly, Kelly);
```

deck.html > html

```
<html>
    <body>
    <script src="deck.js"></script>
</body>
</html>
```

JS deck_test.js > describe("My Function") callback > describe("finalTally") callback > it('finalTally should not succeed if player objects are not valid') callbac

```
var expect = chai.expect;

describe('MyFunction', function() {
    describe('freshDeck', function() {
        it('should create a card deck', function() {
            const testDeck = new Deck();
            expect(testDeck.cards.length).to.equal(52);
        });
        it('numberOfCards property should be defined', function() {
            const testDeck = new Deck();
            expect(testDeck.numberOfCards).to.equal(52);
        });
    });
});



describe("My Function", function() {
    describe("setupGame", function() {
        it("Starts a new game of war with the two players and gives part of the deck to Nixus", function() {
            let Nixus = new Player('Nixus');
            let Adora = new Player('Adora');
            setupGame(Nixus, Adora);
            expect(Nixus.playerDeck).to.be.a('array');
        });
        it('Not start w/o a player 1 or player 2', function() {
            expect(function() {
                setupGame();
            }).to.throw(Error);
        });
    });
});
```

PROMINEO TECH

JS deck_test.js > describe("My Function") callback > describe("finalTally") callback > it('finalTally should not succeed if player objects are not valid') callba

```
35
36  describe("My Function", function() {
37      describe("roundOutput", function() {
38          it('roundOutput should succeed if both player objects are valid and game has been set up', function() {
39              let Nixus = new Player('Nixus');
40              let Adora = new Player('Adora');
41              //need to call game to ensure it runs
42              setupGame(Nixus, Adora);
43              expect(function() {
44                  roundOutput(Nixus, Adora, 1);
45              }).to.not.throw();
46          });
47          it('roundOutput should not succeed if player objects are not valid', function() {
48              let Nixus = new Player('Nixus');
49              let Adora = new Player('Adora');
50              //need to call setupGame properly to ensure the correct error
51              setupGame(Nixus, Adora);
52              expect(function() {
53                  roundOutput(Nixus, null, 1);
54              }).to.throw(Error);
55          });
56      });
57  });
58
59  describe("My Function", function() {
60      describe("playRoundResults", function() {
61          it('Evaluates outcome of each round played with the players and gameSetup', function() {
62              let Nixus = new Player('Nixus');
63              let Adora = new Player('Adora');
64              setupGame(Nixus, Adora);
65              expect(function() {
66                  playRoundResults(Nixus, Adora);
67              }).to.not.throw();
68          });
69          it('With a premade deck it will end with a tie', function() {
70              let Nixus = new Player('Nixus');
71              let Adora = new Player('Adora');
```

PROMINEO TECH

JS deck_test.js > describe("My Function") callback > describe("finalTally") callback > it('finalTally should not succeed if player objects are not valid') callback > expect() callbac

```
describe("My Function", function() {
    describe("playRoundResults", function() {
        it('Evaluates outcome of each round played with the players and gameSetup', function() {
            let Nixus = new Player('Nixus');
            let Adora = new Player('Adora');
            setupGame(Nixus, Adora);
            expect(function() {
                playRoundResults(Nixus, Adora);
            }).to.not.throw();
        });
        it('With a premade deck it will end with a tie', function() {
            let Nixus = new Player('Nixus');
            let Adora = new Player('Adora');
            //changing the amount of cards within the playerDeck
            Nixus.playerDeck = [new Card('diamond', '3'), new Card('heart', 'J'), new Card('club', 'K'), new Card('spade', 'Q')];
            Adora.playerDeck = [new Card('diamond', '4'), new Card('heart', '10'), new Card('club', 'Q'), new Card('spade', 'A')];
            playRoundResults(Nixus, Adora);
            expect(Nixus.playerScore).to.equal(Adora.playerScore);
        });
        it('With a premade deck player 1 /Nixus will win', function() {
            let Nixus = new Player('Nixus');
            let Adora = new Player('Adora');
            //changing the amount of cards within the playerDeck
            Nixus.playerDeck = [new Card('diamond', '3'), new Card('heart', 'J'), new Card('club', 'K')];
            Adora.playerDeck = [new Card('diamond', '4'), new Card('heart', '10'), new Card('club', 'Q')];
            playRoundResults(Nixus, Adora);
            expect(Nixus.playerScore).to.be.above(Adora.playerScore);
        });
        it('With a premade deck player 2 /Adora will win', function() {
            let Nixus = new Player('Nixus');
            let Adora = new Player('Adora');
            //changing the amount of cards within the playerDeck
            Nixus.playerDeck = [new Card('diamond', '3'), new Card('heart', 'J'), new Card('club', '3')];
            Adora.playerDeck = [new Card('diamond', '4'), new Card('heart', '10'), new Card('club', 'Q')];
            playRoundResults(Nixus, Adora);
            expect(Nixus.playerScore).to.be.below(Adora.playerScore);
        });
```

PROMINEO TECH

JS deck_test.js > describe("My Function") callback > describe("finalTally") callback > it('finalTally should not succeed if player objects are not valid') callback > e

```
83              Adora.playerDeck = [new Card('diamond', '4'), new Card('heart', '10'), new Card('club', 'Q')];
84              playRoundResults(Nixus, Adora);
85              expect(Nixus.playerScore).to.be.above(Adora.playerScore);
86          });
87          it('With a premade deck player 2 /Adora will win', function() {
88              let Nixus = new Player('Nixus');
89              let Adora = new Player('Adora');
90              //changing the amount of cards within the playerDeck
91              Nixus.playerDeck = [new Card('diamond', '3'), new Card('heart', 'J'), new Card('club', '3')];
92              Adora.playerDeck = [new Card('diamond', '4'), new Card('heart', '10'), new Card('club', 'Q')];
93              playRoundResults(Nixus, Adora);
94              expect(Nixus.playerScore).to.be.below(Adora.playerScore);
95          });
96      });
97  });
98
99  describe("My Function", function() {
100     describe("finalTally", function() {
101         it('finalTally should succeed if both player objects are valid and game has been set up', function() {
102             let Nixus = new Player('Nixus');
103             let Adora = new Player('Adora');
104             setupGame(Nixus, Adora);
105             expect(function() {
106                 finalTally(Nixus, Adora);
107             }).to.not.throw();
108         });
109         it('finalTally should not succeed if player objects are not valid', function() {
110             let Nixus = new Player('Nixus');
111             let Adora = new Player('Adora');
112             //need to call setupGame properly to ensure the correct error
113             setupGame(Nixus, Adora);
114             expect(function() {
115                 finalTally(Nixus, null, 1);
116             }).to.throw(Error);
117         });
118     });
119 });
```

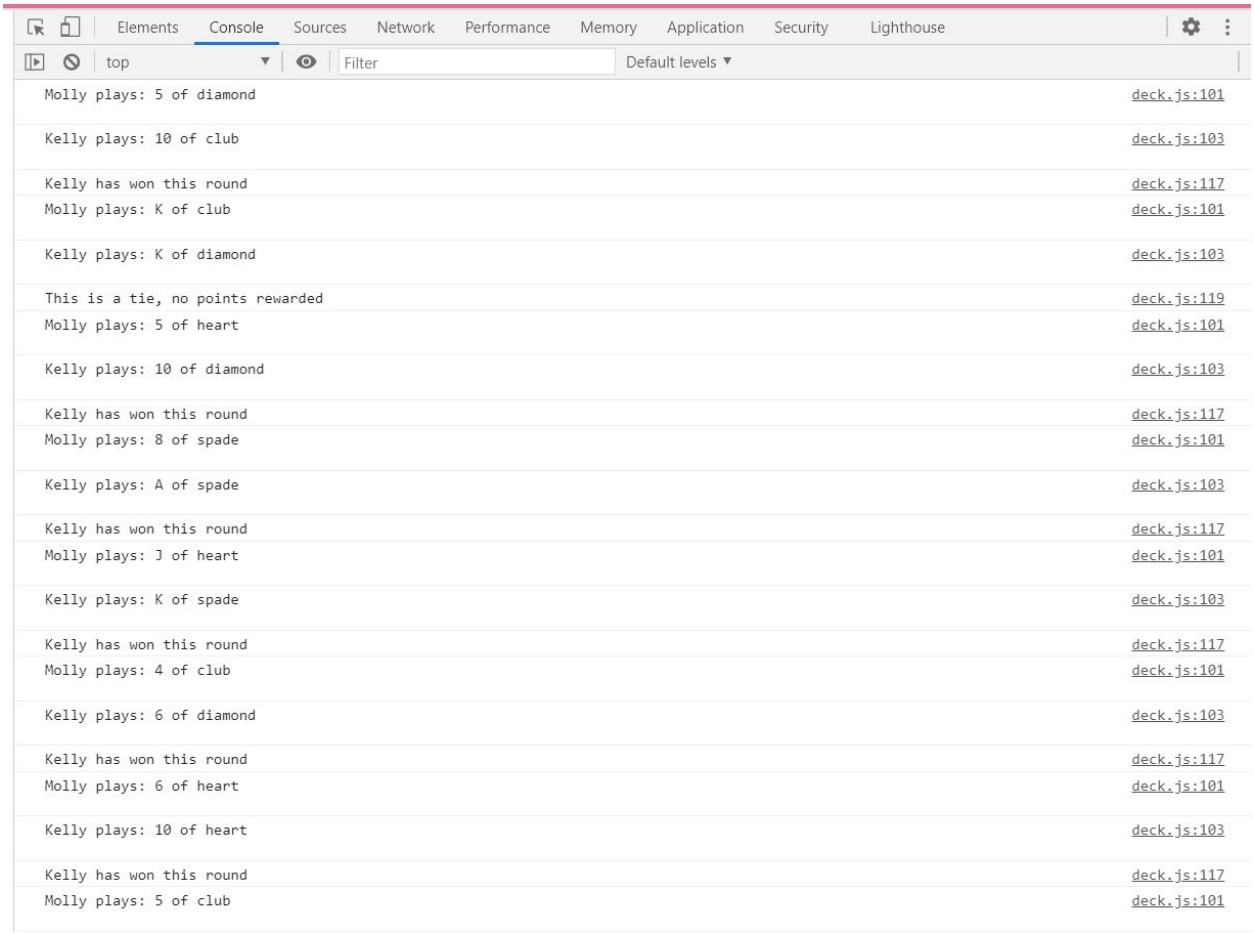
deck_test.html > html > head

```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <link rel="stylesheet" href="node_modules/mocha/mocha.css">
5       </head>
6       <body>
7           <div id="mocha"><p><a href="."></a></p></div>
8           <div id="messages"></div>
9           <div id="fixtures"></div>
10          <script src="node_modules/mocha/mocha.js"></script>
11          <script src="node_modules/chai/chai.js"></script>
12          <script src="deck.js"></script>
13          <script>mocha.setup('bdd')</script>
14          <script src="deck_test.js"></script>
15          <script>mocha.run();</script>
16      </body>
17  </html>
```

PROMINEO TECH

PROMINEO TECH

Screenshots of Running Application:

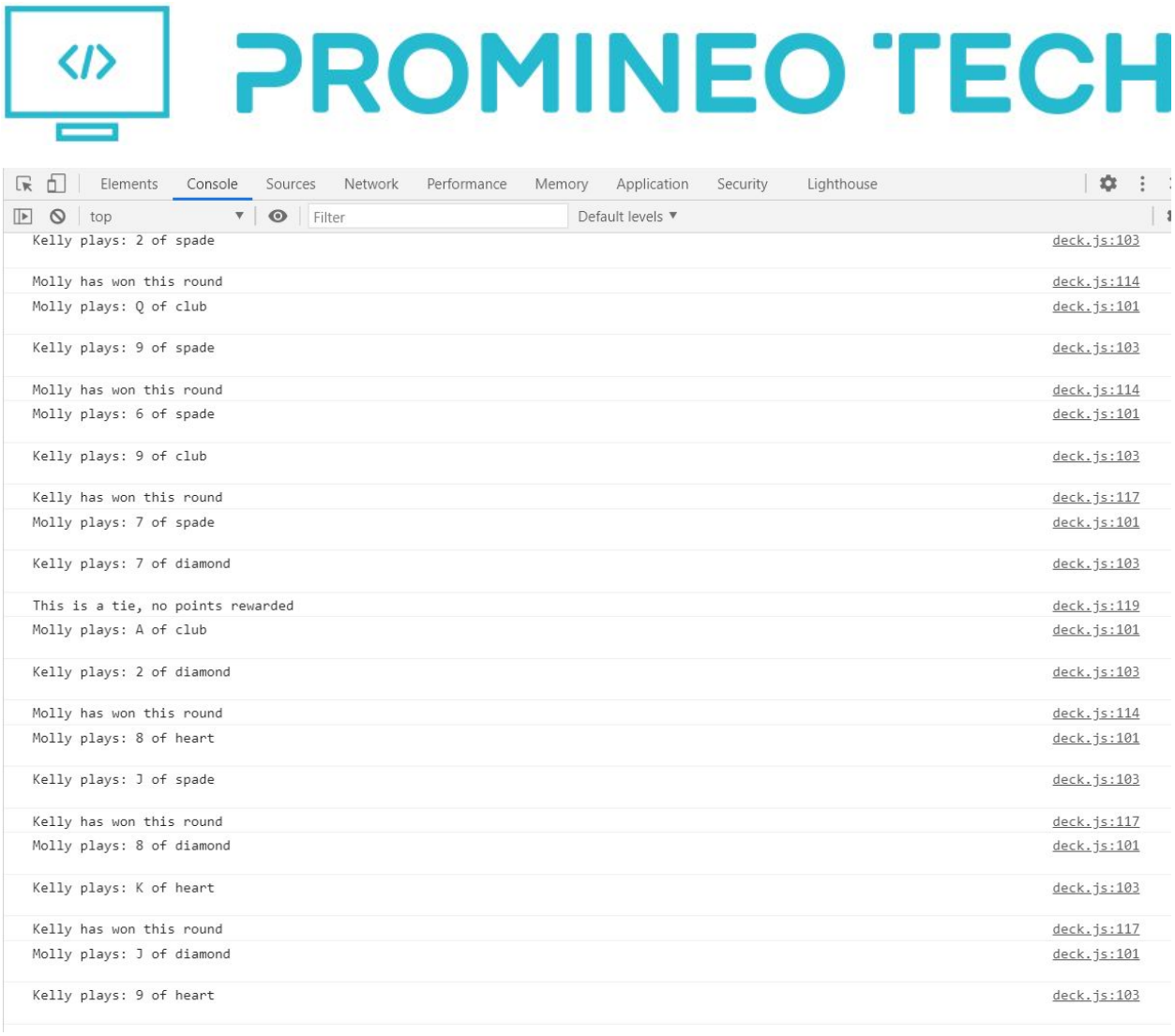


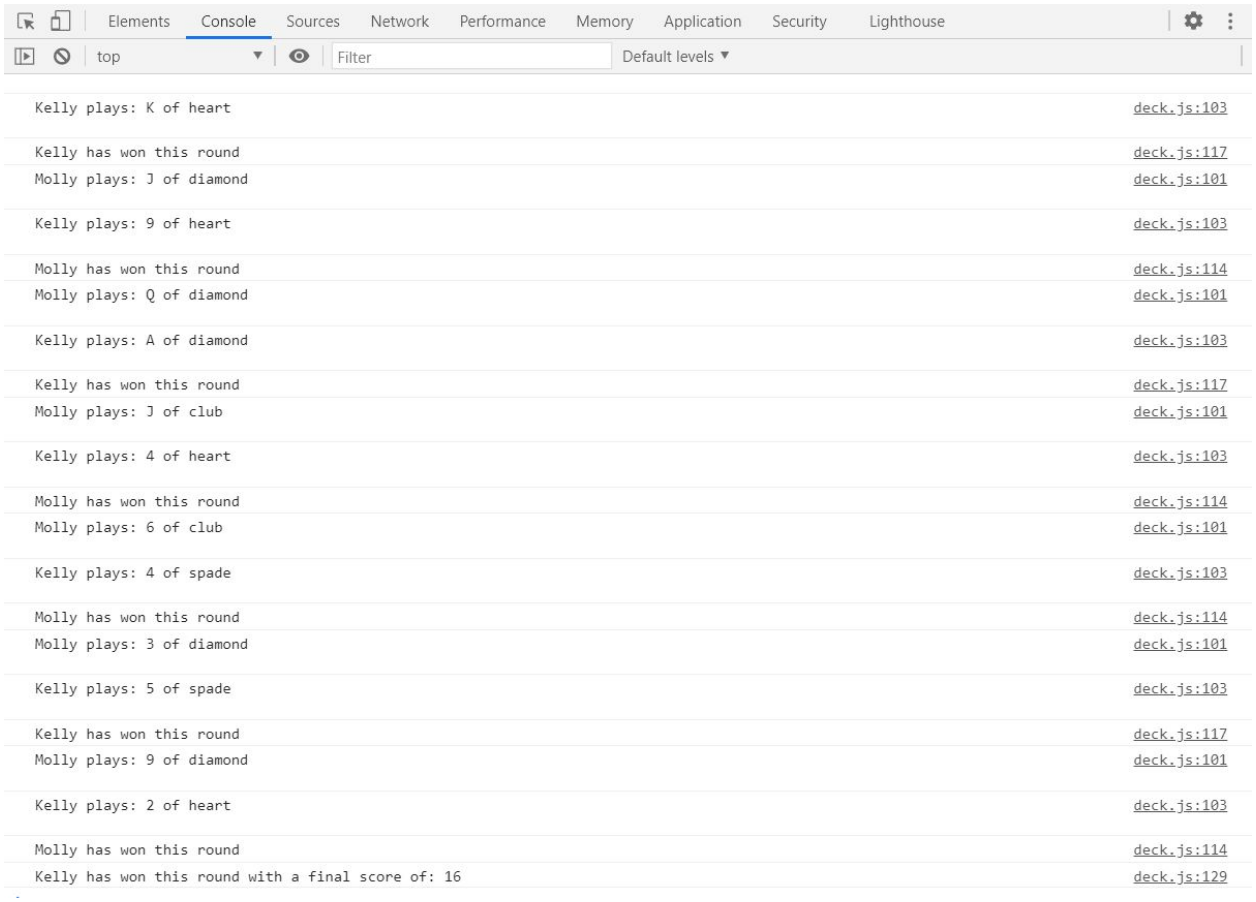
PROMINEO TECH

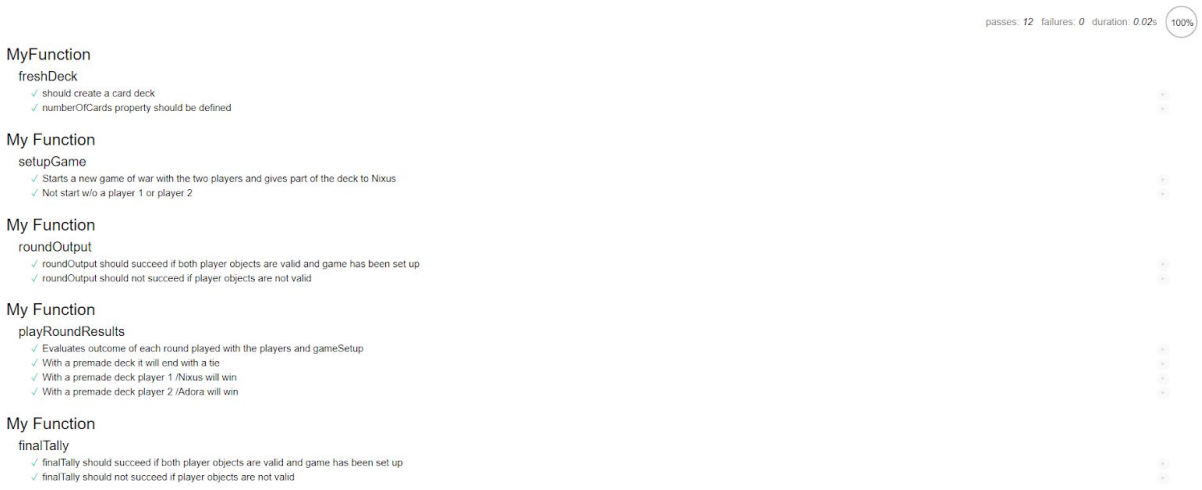
Elements Console Sources Network Performance Memory Application Security Lighthouse

top Filter Default levels

Kelly has won this round	deck.js:117
Molly plays: 5 of club	deck.js:101
Kelly plays: 10 of spade	deck.js:103
Kelly has won this round	deck.js:117
Molly plays: 2 of club	deck.js:101
Kelly plays: 4 of diamond	deck.js:103
Kelly has won this round	deck.js:117
Molly plays: 3 of spade	deck.js:101
Kelly plays: 8 of club	deck.js:103
Kelly has won this round	deck.js:117
Molly plays: 3 of heart	deck.js:101
Kelly plays: A of heart	deck.js:103
Kelly has won this round	deck.js:117
Molly plays: 3 of club	deck.js:101
Kelly plays: 7 of club	deck.js:103
Kelly has won this round	deck.js:117
Molly plays: Q of spade	deck.js:101
Kelly plays: Q of heart	deck.js:103
This is a tie, no points rewarded	deck.js:119
Molly plays: 7 of heart	deck.js:101
Kelly plays: 2 of spade	deck.js:103
Molly has won this round	deck.js:114


PROMINEO TECH
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
top
Filter
Default levels
Kelly plays: 2 of spade deck.js:103
Molly has won this round deck.js:114
Molly plays: Q of club deck.js:101
Kelly plays: 9 of spade deck.js:103
Molly has won this round deck.js:114
Molly plays: 6 of spade deck.js:101
Kelly plays: 9 of club deck.js:103
Kelly has won this round deck.js:117
Molly plays: 7 of spade deck.js:101
Kelly plays: 7 of diamond deck.js:103
This is a tie, no points rewarded deck.js:119
Molly plays: A of club deck.js:101
Kelly plays: 2 of diamond deck.js:103
Molly has won this round deck.js:114
Molly plays: 8 of heart deck.js:101
Kelly plays: J of spade deck.js:103
Kelly has won this round deck.js:117
Molly plays: 8 of diamond deck.js:101
Kelly plays: K of heart deck.js:103
Kelly has won this round deck.js:117
Molly plays: J of diamond deck.js:101
Kelly plays: 9 of heart deck.js:103





URL to GitHub Repository:

https://github.com/JoleneMel/Week-Six-Assignment