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.

**Screenshots of Code:**

//war.js for FESD Javascript Week6 - Final Profect - The Card Game WAR

//By:  Tim Gibney - FESD 11/2020

// 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.

class Card {

    constructor(value,suit){

        this.value = value;

        this.suit = suit;

    }

    listCard(){

        console.log(`The card is ${this.value}${this.suit}`);

    }

}

//A container class for card objects

class CardDeck {

    constructor(){

        this.cards = [];

    }

    //Called from program start

    buildDeck(){ //card values will be 1 - 13 for A - K respectively

        for(var i = 0;i < 13;i++){

            this.cards[i] = new Card(i + 1,"H");

            this.cards[i + 13]= new Card(i + 1,"S");

            this.cards[i + 26]= new Card(i + 1,"D");

            this.cards[i + 39]= new Card(i + 1,"C");

        }

    }

    //shuffleDeck takes the cards array and shuffles it using the Durstenfeld Shuffle algorithm

    //which acts upon the array passed and changes/shuffles the array/deck in-place

    //Called from program start

    shuffleDeck(){

        for (let i = this.cards.length - 1; i > 0; i--) {

            const j = Math.floor(Math.random() \* (i + 1));

            [this.cards[i], this.cards[j]] = [this.cards[j], this.cards[i]];

        }

    }

    listDeck(){

        console.log(this.cards);

    }

}

class Player {

    constructor(firstName) {

        this.firstName = firstName;

        this.handCards = [];

        this.wonCards = [];

        this.score = 0;

    }

    listPlayerCards(){

        console.log(this.handCards);

    }

}

class Game {

    constructor(name){

        this.cardGameName = name;

        this.numberOfPlayers = 2;

        this.numberOfCardsDealt = 26;

        this.tableCards = [];

        this.rounds = 0;

        this.players = [];

    }

    //Called from .playGame()

    addPlayer(playerName){

        this.players.push(new Player(playerName));

    }

    //dealCards uses .pop to take the last card from the deck array and uses .push to place that card

    //into the player's hand array at the first element for the first hand or the last element for each

    //successive hand

    //Called from program start

    dealCards(){

        for(let i = 0;i < this.numberOfCardsDealt;i++) {

            for (let p = 0; p < this.numberOfPlayers; p++){

                this.players[p].handCards.push(theDeck.cards.pop());

            }

        }

    }

    //playHand takes a players handcards array and .pops that array's last card on the tableCards array

    //in preparation for analysis/scoring/handedness

    //called from .playGame()

    playHand(player){

        this.tableCards.push(player.handCards.pop());

    }

    //winnerTakesTheCards takes the two cards played into tableCards and .pops them from tableCards

    //stack and pushes them into the winning player's hand  and a

    //point is applied appropriately or a tie occurs in which case the cards go

    //to the bit bucket

    //Called from .playGame()

    winnerTakesTheCards(winner){

        if(winner == 1){

            for(var i = 0; i < this.tableCards.length; i++) {

                this.players[0].wonCards.push(this.tableCards.pop());

            }

        }else if(winner == 2){

            for(var i = 0; i < this.tableCards.length; i++) {

            this.players[1].wonCards.push(this.tableCards.pop());

            }

        }else if(winner == "tie"){

            for(var i = 0; i < this.tableCards.length; i++) {

            this.tableCards.pop();

            }

        }

    }

    //Called from .playGame()

    displayScore(){

        for (var i = 0; i < this.players.length; i++)

        console.log(

            `

            ${this.players[i].firstName} has ${this.players[i].score} points.

            `

        );

    }

    //option when Ace and King are played simultaneously

    //Called from .playGame()

    aceTrumpsKing() {

        if(this.tableCards[0] == 1 && this.tableCards[1] == 13) {

            this.players[0].score += 1;

            this.winnerTakesTheCards(1);

            return true;

        } else if(this.tableCards[0] == 13 && this.tableCards[1] == 1) {

            this.players[1].score += 1;

            this.winnerTakesTheCards(2);

            return true;

        } else {

            return false;

        }

    }

    //Each player pops their top card onto table and rounds is incremented

    //aceTrumpsKing() evaluates the cards on the table initially checking if

    //an Ace and King have been played simultaneously and if so applies a point

    //accordingly.  Otherwise play is passed back to complete the loop and

    //results are passed to .winnerTakesTheCards().

    //Run after .dealCards()

    playGame(){

        while(this.players[0].handCards.length != 0 && this.players[1].handCards.length != 0){

            this.playHand(this.players[0]);

            this.playHand(this.players[1]);

            this.rounds += 1;

            if(!this.aceTrumpsKing()) {

                if(this.tableCards[0].value > this.tableCards[1].value){

                    this.players[0].score += 1;

                    this.winnerTakesTheCards(1);

                }else if(this.tableCards[0].value < this.tableCards[1].value){

                    this.players[1].score += 1;

                    this.winnerTakesTheCards(2);

                }else{

                    this.winnerTakesTheCards("tie");

                }

            }

        }

        this.displayScore();

    }

}

//Program start

theDeck = new CardDeck();

theDeck.buildDeck();

theDeck.shuffleDeck();

War = new Game("War");

War.addPlayer("Player1");

War.addPlayer("Player2");

War.dealCards();

War.playGame();

<!-- war.html -->

<html>

    <head>

        <script src="war.js"></script>

        <meta charset="utf-8">

        <title>FESD Week 6 - Final Project - The Card Game WAR with Unit Test</title>

    </head>

    <body>

        <p>For the final project you will be creating an automated version of the classic card game WAR.</p>

        <p>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:</p>

        <p><ol>

        <li>-   Deal 26 Cards to two Players from a Deck. </li>

        <li>-   Iterate through the turns where each Player plays a Card</li>

        <li>-   The Player who played the higher card is awarded a point</li>

        <li>-   Ties result in zero points for either Player</li>

        <li>-   After all cards have been played, display the score.</li>

        </ol></p>

        <p>

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

        </p>

    </body>

</html>

//war\_test.js

var expect = chai.expect;

describe("MyFunctions", function() {

    describe("buildDeck", function() {

        it("should create a card deck", function() {

            testDeck = new CardDeck; //follow the process that is used in the main .js

            testDeck.buildDeck();    //to get to the point of test

            expect(testDeck.cards.length).to.equal(52);

        });

        it("should create each card as an instance of Card", function() {

            testDeck = new CardDeck;

            testDeck.buildDeck();

            expect(testDeck.cards[51]).to.be.an.instanceOf(Card);

        });

        it("should throw an error if card deck is not created", function() {

            expect(function() {

                testDeck = new CardDeck;

                testDeck.buildDeck();

                expect(testDeck.cards.length).to.equal(0);

            }).to.throw(Error);

        });

    });

    describe("addPlayer", function() {

        it("should create a new Player in Game.players[]", function() {

            testGame = new Game;

            testGame.addPlayer("testPlayer1");

            expect(testGame.players[0].firstName).to.equal("testPlayer1");

        });

        it("should throw and error if testGame.players[1].name is undefined or null", function() {

            expect(function() {

                testGame = new Game;

                testGame.addPlayer("testPlayer");

                expect(testGame.players[1].name).to.equal("undefined");

            }).to.throw(Error);

        })

    })

    describe("dealCards", function() {

        it(`should populate 26 cards into Game.players[0] (&[1]) .handcards`, function() {

            testDeck = new CardDeck;

            testDeck.buildDeck();

            testDeck.shuffleDeck();

            testGame = new Game("War");

            testGame.addPlayer("testPlayer1");

            testGame.addPlayer("testPlayer2");

            testGame.dealCards();

            expect(testGame.players[0].handCards.length).to.equal(26);

            expect(testGame.players[1].handCards.length).to.equal(26);

        });

        it(`should throw and error if cards have not been dealt

             from the deck to the player's hand`, function() {

            expect(function() {

                testDeck = new CardDeck;

                testDeck.buildDeck();

                testDeck.shuffleDeck();

                testGame = new Game("War");

                testGame.addPlayer("testPlayer1");

                testGame.addPlayer("testPlayer2");

                testGame.dealCards();

                expect(testGame.players[0].handCards.length).to.equal(0);

                expect(testGame.players[1].handCards.length).to.equal(0);

            }).to.throw(Error);

        })

    })

});

<!--tests.html-->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <link rel="stylesheet" href="node\_modules/mocha/mocha.css">

</head>

<body>

    <div id="mocha"><p><a href="."></a></p></div>

    <div id="messages"></div>

    <div id="fixtures"></div>

    <script src="node\_modules/mocha/mocha.js"></script>

    <script src="node\_modules/chai/chai.js"></script>

    <script src="war.js"></script>

    <script>mocha.setup("bdd");</script>

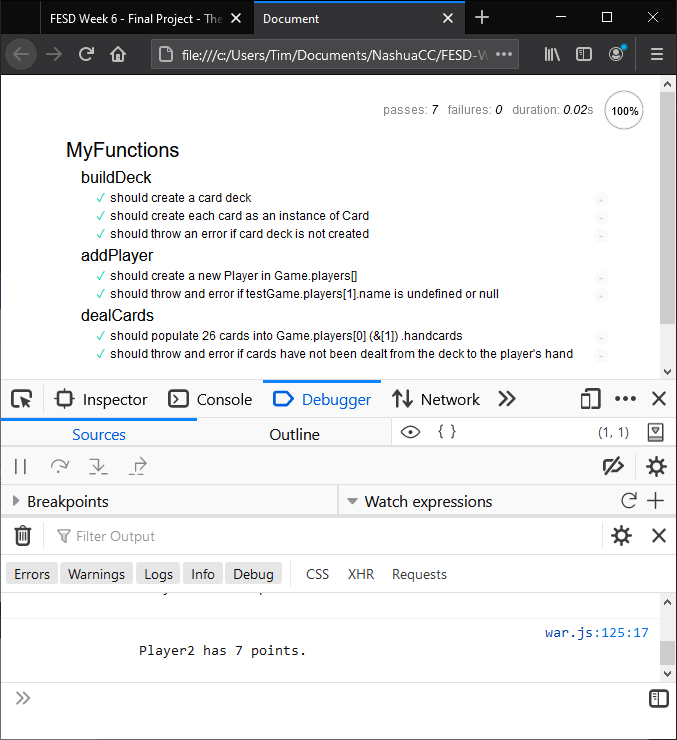
    <script src="war\_test.js"></script>

    <script>mocha.run();</script>

</body>

</html>

**Screenshots of Running Application:**



**URL to GitHub Repository:**

**https://github.com/mctimoth/FESD-Week6-WARWithUnitTest**