Jonathan West

05/11/2018

**Test Cases**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Object State/Comments** | **Expected Result** |
| Deck testDeck = new Deck(); | Size = 52  Front = 2S  Rear = Slot after KH |  |
| testDeck.size(); |  | 52 |
| testDeck.deal(); | Size = 51  Front = 3S  Rear = Slot after KH | 2S |
| testDeck.deal(); | Size = 50  Front = 4S  Rear = Slot after KH | 3S |
| testDeck.size(); |  | 50 |

|  |  |  |
| --- | --- | --- |
| **Operation** | **Object State/Comments** | **Expected Result** |
| Deck testDeck = new Deck(); | Size = 52  Front = 2S  Rear = Slot after KH |  |
| testDeck.shuffle(); | Size = 52  Front = random  Rear = slot after last card |  |
| testDeck.deal(); | Size = 51  Front = different random  Rear = slot after last card | Random card |
| testDeck.size(); |  | 51 |
| Repeat this test case multiple times to ensure that cards are random. |  |  |

|  |  |  |
| --- | --- | --- |
| **Operation** | **Object State/Comments** | **Expected Result** |
| War game = new War(); | drawDeck.size = 52  Player1.hand.size = 0  Player2.hand.size = 0  drawDeck is unshuffled |  |
| game.start(); | drawDeck.size = 0  Player1.hand.size = 26  Player2.hand.size = 26  drawDeck is shuffled before cards are dealt to players |  |
| game.play();  Play through game to make sure cards are random |  | The Player with the higher ranked card |

|  |  |  |
| --- | --- | --- |
| **Operation** | **Object State/Comments** | **Expected Result** |
| Card[] war1 = {“2S”, “2H”, “3S”, “4S”, “5S”, “3H”, “4H”, “5H”};  Card[] war2 = {“6S”, “6H”, “7S”, “8S”, “9S”, “7H”, “8H”, “9H”}; | Just to get some set arrays of cards |  |
|  |  |  |
| War game = new War(); | drawDeck = 52  Player1.hand.size = 0  Player2.hand.size = 0  drawDeck is unshuffled |  |
| game.getDrawDeck(); | If testing is not enabled, don’t allow public access to deck. | Exception (Some sort of inaccessible method exception) |
| game.enableTesting(true); |  | True |
| game.getDrawDeck(); |  | drawDeck |
| game.getDrawDeck().sort(war1, “6S”, “10S”); | Set up deck so that 1 war will occur  The way sort works is that it adds cards with the ranks and suits specified in the argument strings (Checking whether there are duplicates) to a new ListQueue of cards. Once the argument list ends, it adds remaining cards of a standard deck to the deck (Makes sure there are no duplicates) |  |
| game.start() | drawDeck = 0  Player1.hand.size = 26  Player2.hand.size = 26  The cards are dealt from the deck one player at a time (1st card goes to player1, 2nd card goes to player2, 3rd card goes to player1, etc)  When testing is enabled, start does not shuffle the deck. |  |
| game.play(); | Kitty.size = 2  Player1 played card = 2S  Player2 played card = 2H | War! |
| tie() (This is called in game.play() when a tie happens) | kitty.size = 8  Player1.hand.size = 22  Player1.hand.size = 22 |  |
| game.play() | Kitty.size = 10  Player1 played card = 6S  Player2 played card = 10S | Player2 wins the hand |
| ELABORATION OF PREVIOUS ACTION | Player1.hand.size = 21  Player2.hand.size = 31 | Player2 gets the cards in kitty, Player1’s played card and their own played card (Both are added to the kitty between being played and being picked up by the winner or a tie) |
|  |  |  |
| War game = new War();  game.enableTesting(true); |  |  |
| game.getDrawDeck().sort(war1, war2, “KS”, “QH”); | Test for multiple wars |  |
| Game is played until end of second war is reached… (Same process as single war) | Kitty.size = 16  Player1.hand.size = 18  Player2.hand.size = 18 |  |
|  | Kitty.size = 18  Player1 played card = KS  Player2 played card = QH | Player1 wins the hand |
|  | Kitty.size = 0  Player1.hand.size = 35  Player2.hand.size = 17 |  |

|  |  |  |
| --- | --- | --- |
| **Operation** | **Object State/Comments** | **Expected Result** |
| War game = new War(); |  |  |
| Configure deck so that game ends because Player2 had 1 last card that was lower than Player1’s next card |  |  |
|  | Player1 played card = KS  Player2 played card = JC | Player1 wins |

|  |  |  |
| --- | --- | --- |
| **Operation** | **Object State/Comments** | **Expected Result** |
| War game = new War(); |  |  |
| Configure deck so that Player2 ends up with 1 card left, but will win the round. |  |  |
|  | Player1 played card = 2S  Player2 played card = AS | Game will continue |

|  |  |  |
| --- | --- | --- |
| **Operation** | **Object State/Comments** | **Expected Result** |
| War game = new War(); |  |  |
| Configure deck so that Player2 will have only 2 cards left and a tie will occur. Player2 loses the game because they do not have enough cards to put to the kitty. |  |  |
| game.play() | Player1 hand size = 49  Player2 hand size = 3 |  |
|  | Kitty size = 2  Player1 played card = “JS”  Player2 played card = “JH”  Player1 hand size = 48  Player2 hand size = 2 | War! |
|  |  | Player2 does not have enough cards to engage in war. Player1 wins! |