**Test Cases for Deck Class**

**Test Case 1: createDeck() : void**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The method must set the unshuffledDeck variable to a list of 52 cards | 1. 52 | 1. Everything else |
| * The "cards" in the unshuffledDeck queue are instances of the Card class | 1. True | 1. False |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | unshuffledDeck.size() = 52 |
| 2 | unshuffledDeck.size() = 51 |
| 3 | unshuffledDeck.peek() instanceof Card = true |
| 4 | unshuffledDeck.peek() instanceof Card = false |

**Test Case 2: shuffle() : void**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The shuffled deck (result) must be different from the unshuffled one | 1. They are different | 1. They are the same |
| * Both the shuffled and unshuffled decks must have a size of 52 | 1. 52 | 1. Numbers other than 52 |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | Shuffled deck: [AC, KC, QC, JC, 10C …]  Unshuffled deck: [5D, 4C, KS, 9H, 9D …] |
| 2 | Shuffled deck: [AC, KC, QC, JC, 10C …]  Unshuffled deck: [AC, KC, QC, JC, 10C …] |
| 3 | Shuffled deck with a size 52  Unshuffled deck with a size of 52 |
| 4 | Unshuffled deck with a size of 52  Shuffled deck with a size of 51 |
| 4 | Unshuffled deck with a size of 52  Shuffled deck with a size of 53 |

**Test Case 3: deal(): Card**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The method must remove the first card from a deck | 1. Removes the first card from a deck | 1. Removes a card other than the first one |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | deck: AD, 4C, 3H  deck.deal() : AD |
| 2 | deck: AD, 4C, 3H  deck.deal() : 4C |

**Test Case 4: size(): int**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * Must Return the size of the number of cards that have not been dealt yet | 1. Returns the correct size of shuffledDeck | 1. Returns the wrong size of shuffledDeck |
| * Must Return a value between 0 and 52 (inclusive) | 1. 0 <= size <= 52 | 1. size < 0 OR size > 52 |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1,3 | shuffledDeck: 52 cards  deck.size() = 52 |
| 1,3 | shuffledDeck: 0 cards  deck.size() = 0 |
| 1,3 | shuffledDeck: 51 cards  deck.size() = 51 |
| 1,3 | shuffledDeck: 1 card  deck.size() = 1 |
| 2,4 | shuffledDeck: 1 card  deck.size() = -1 |
| 2,4 | shuffledDeck: 52 cards  deck.size() = 53 |
| 2,4 | shuffledDeck: 52 cards  deck.size() = 54 |

**Test Cases for Card Class**

**Test Case 1: toString(): String**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * Returns a string of the rank and the suit of the current Card object | 1. Returns the correct string | 1. Returns the wrong string |
| * The object must contain the right properties | 1. the object has a rank, suit and value in the right order and with correct type of values | 1. Invalid object |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1,3 | Card: {suit: Spades, rank: 10, value: 9} |
| 2 | Card: {suit: Hearts, rank: 2, value: 1} |
| 4 | Card: {suit: 2, rank: Spades, value: 1} |

**Test cases for Player Class**

**Test Case 1: cardsLeft(): int**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * Must return the number of cards left in the player's hand | 1. Returns the correct number of cards | 1. Returns the wrong number of cards |
| * The returned number must be 0 <= size <= 26 | 1. 0 <= size <= 26 | 1. size < 0 OR size > 26 |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1,3 | playerHand: [AC, 2D, 3S]  number of cards: 3  cardsLeft () = 3 |
| 2,3 | playerHand: [AC, 2D, 3S]  number of cards: 3  cardsLeft () = 2 |
| 2,4 | playerHand: [AC, 2D, 3S, … ]  number of cards: 26  cardsLeft () = 27 |

**Test case 2: getFirstCard() : Card**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * Returns the first card object from the player's hand | 1. returns the first Card | 1. Else |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | playerHand: [Card:2C, Card:JH, Card:5D, Card:7C, Card:10S]  getFirstCard().toString() = "2 of Clubs" |
| 2 | playerHand: [Card:2C, Card:JH, Card:5D, Card:7C, Card:10S]  getFirstCard().toString() = "Jack of Hearts"  playerHand: ["", Card:JH, Card:5D, Card:7C, Card:10S]  getFirstCard().toString() = NotACardException() |

**Test case 2 : addCards(cards: Queue<Card>) : boolean**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The Queue that is passed to the method cannot be empty | 1. isEmpty() = false | 1. isEmpty() = true |
| * The method must add the cards to the end of the player's hand | 1. The last item added to the hand must be the last in the hand itself | 1. The last item in hand is not the last item of the cards Queue past into the method |
| * The number of cards inside the hand shouldn't exceed 52 | 1. 52 | 1. Other |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | cards: [Card:2C, Card:JH, Card:5D, Card:7C, Card:10S] |
| 2 | cards: [] |
| 3 | cards: [Card:2C, Card:JH, Card:5D, Card:7C, Card:10S] |
| 4 | cards: [Card:2C, Card:JH, Card:5D, Card:7C, Card:10S] |
| 5 | cards: [2 cards], playerHand: [25 cards] |

**Test Cases for War Class**

**Test Case 1: start(): boolean**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * To start, the deck must have 52 cards inside | 1. true | 1. false |
| * At the end, each player must have 26 cards in order to start the game | 1. true | 1. false |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | deck: [52 card objects] |
| 2 | deck: [26 card objects],  deck: [51 card objects],  deck: [53 cards objects] |
| 3 | player 1: 26 cards  player 2: 26 cards |
| 4 | player 1: 26 cards  player 2: 25 cards  player 1: 25 cards  player 2: 26 cards  player 1: 27 cards  player 2: 26 cards  player 1: 26 cards  player 2: 27 cards |

**Test Case 2: isWar(): boolean**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The method must check the ranks of the played cards | 1. Ace, King, Queen, Jack, 10, 9, 8, 7, 6, 5, 4, 3, 2 | 1. Other |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | Card 1 rank: Ace  Card 2 rank: Ace, |
| 2 | Card 1 rank: 2  Card 2 rank: 3 |

**Test Case 3: isGameOver(): boolean**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The game is over when one of the players doesn't have any cards left | 1. A player with an empty hand | 1. No player with an empty hand |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | Player 1: [0 cards left]  Player 2: [52 cards left] |
| 2 | Player 1: [1 card left]  Player 2: [51 cards left] |

**Test Case 4: isWin(): boolean**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The round is won when one player has a higher rank card than the other player | 1. The ranks are different | 1. The ranks are the same |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | Player 1 card rank: Ace  Player 2 card rank: King,  Player 1 card rank: 3  Player 2 card rank: 2, |
| 2 | Player 1 card rank: Ace  Player 2 card rank: Ace, |

**Test Case 5: playWar(): void**

**Rules:**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The players must return 4 cards | 1. 4 | 1. Other |
| * The method must check for a loss (a player is left without any cards) | 1. no cards left | 1. can still play |
| * Must check if there's another war or not | 1. It's a war | 1. it's a win |

**Mapping:**

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1,2,4 | Player 1 hand: [5 cards left]  Player 2 hand: [47 cards left] |
| 1,2,3 | Player 1 hand: [4 cards left]  Player 2 hand: [48 cards left] |
| 1,4,5 | Player 1 hand: [Card:2C, Card:JH, Card:5D, Card:7C, Card:10S, …]  Player 2 hand: [Card:2C, Card:JH, Card:5D, Card:7C, Card:10S, …] |
| 1,4,6 | Player 1 hand: [Card:2C, Card:JH, Card:5D, Card:6C, Card:10S, …]  Player 2 hand: [Card:2C, Card:JH, Card:5D, Card:7C, Card:10S, …] |