14/04/2025, 12:33 PlayerTest

```
package Game;
        import org.junit.jupiter.a
import org.junit.jupiter.a
import java.util.HashSet;
import java.util.Set;

                           import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.util.HashSet;
import java.util.Set;

import static org.junit.jupiter.api.Assertions.*;

class DeckTest {

    private Deck deck;

    @BeforeEach
    void setUp() {
        deck = new Deck();
    }

    aTest
    void testInitialDeckSize() {
        int count = 0;
        while (!deck.isEmpty()) {
            deck.draw();
            count++;
        }

        assertEquals(52, count, "Deck should have 5
    }

    int count = 0;
    while (!deck.isEmpty()) {
            deck.draw(), "Draw should }
    }

    int count = 0;
    while (!deck.isEmpty()) {
            deck.draw();
            count++;
    }

        assertFquals(52 - draws, count, "Deck should }
    }

    int count = 0;
    while (!deck.isEmpty()) {
            deck.draw();
            count++;
    }

    assertFquals(52 - draws, count, "Deck should be assertFquals(62ck.isEmpty(), "Deck should should assertFalse(deck.draw(), "Card should assertFountWill(deck.draw(), "Card should assertNouNull(deck.draw(), "Card should be assertNull(deck.draw(), "Draw from empty destinated be assertNull(deck.draw(), "Draw from empty destinated be assertNull(deck.draw(), "Draw from empty destinated be assertNull(deck.draw();
        assertFquals(52, uniqueCards = new HashSet ◇();
    while (!deck.isEmpty()) {
        Card card = deck.draw();
        assertFquals(52, uniqueCards.size(), "All colored be assertEquals(52, uniqueCards.size(), "All colored beauty assertEquals(52, uniqueCards.size(), "All colored beauty assertEquals(52, uniqueCards.size(), "All colored beauty assertEqua
                                                                     assertEquals(52, count, "Deck should have 52 cards initially");
                                                                 int draws = 5;
for (int i = 0; i < draws; i++) {
    assertNotNull(deck.draw(), "Draw should return a card");
}</pre>
                                                                     {\tt assertEquals} ({\tt 52 - draws, count, "Deck should have 52 - draws cards left"});\\
                                                 woid testIsEmptyAfterAllDraws() {
   for (int i = 0; i < 52; i++) {
        assertFalse(deck.isEmpty(), "Deck should not be empty before 52 draws");
        assertNotNull(deck.draw(), "Card should not be null before deck is empty");
}</pre>
                                                                   assertTrue(deck.isEmpty(), "Deck should be empty after 52 draws"); assertNull(deck.draw(), "Draw from empty deck should return null");
                                             aTest

void testAllCardsAreUnique() {

Set<String> uniqueCards = new HashSet♦();
while (!deck.isEmpty()) {

Card card = deck.draw();
assetNotMull(card, "Card should not be null");
uniqueCards.add(card.toString());
}
                                                                      assertEquals(52, uniqueCards.size(), "All cards in the deck should be unique");
```