File - C:\Users\niall\IdeaProjects\B00787301_COM498_Assignment2\src\Board.java

```java
import java.util.Arrays;
import java.util.Scanner;

/**
 * This class holds the Board functionality of the card game such as
 * the BOARD listArray and methods to manipulate the Cards in BOARD.
 * It also contains the Replay Queue functionality.
 */

public class Board
{
    protected Deck deck;
    protected final AListArray<Card> BOARD = new AListArray<Card>();
    private final Queue<String> REPLAY = new Queue<String>();
    private static final int BOARDNUMCARDS = 9;

    public Board() // board constructor creates new deck instance and deals 9 cards
    {
        deck = new Deck();
        for (int i = 0; i < BOARDNUMCARDS; i++) {
            BOARD.add(deck.deal());
        }
    }

    public void newBoard() // resets the board ListArray and replay queue and deals 9 cards from new deck
    {
        BOARD.clear();
        REPLAY.clear();
        this.deck = new Deck();
        for (int i = 0; i < BOARDNUMCARDS; i++) {
            BOARD.add(this.deck.deal());
        }
    }

    public int getBoardLength() // returns length of board
    {
        return BOARD.getLength();
    }

    public Card getBoardEntry(int index) // returns specific card entry from board
    {
        return BOARD.getEntry(index);
    }


    protected void displayBoard(AListArray<Card> CHECK) // displays the cards on the board to user
    {
        System.out.println("\n-----------Board-------------");
        for (int i = CHECK.getLength(); i > 0; i--)
            {
            System.out.println("Slot " + (i) + "  -  " + CHECK.getEntry(i));
            }
        System.out.println("----------------------------");
        System.out.println(deck.getDeckLength() + " cards left in deck.");
        System.out.println("----------------------------\n");
    }

    protected void replaceCards(int replace1, int replace2)
{                               // replaces the cards in slot numbers passed in or removes them if deck is empty
        REPLAY.enqueue("Board: " + BOARD.toString());
        int[] replace = {replace1,replace2};
        Arrays.sort(replace);
        for(int i = 1; i >= 0; i--) {
            REPLAY.enqueue("Removed: "+ BOARD.getEntry(replace[i]).toString());
            if (deck.getDeckLength() > 0) {
                BOARD.replace(replace[i], deck.deal());
            }
            else {
                BOARD.remove(replace[i]);
            }
        }

    }

```

```
 75     protected void replaceCards(int replace1, int replace2, int replace3)
 76 {                               // replaces the cards in slot numbers passed in or removes them if
    deck is empty
 77         REPLAY.enqueue("Board: " + BOARD.toString());
 78         int[] replace = {replace1,replace2,replace3};
 79         Arrays.sort(replace);
 80         for(int i = 2; i >=0; i--) {
 81             REPLAY.enqueue("Removed: "+ BOARD.getEntry(replace[i]).toString());
 82             if (deck.getDeckLength() > 0) {
 83                 BOARD.replace(replace[i], deck.deal());
 84             }
 85             else {
 86                 BOARD.remove(replace[i]);
 87             }
 88         }
 89     }
 90
 91     protected AListArray<Integer> checkPossibleMoves(AListArray<Card> CHECK)
 92 {                               // checks for any possible moves in the board passed, returns slot
    numbers as HINT listArray if found, otherwise empty
 93         AListArray<Integer> HINT = new AListArray<Integer>();
 94         for(int i = 1; i <= CHECK.getLength(); i++) {
 95             for (int j = i + 1; j <= CHECK.getLength(); j++) {
 96                 if (CHECK.getEntry(i).equalEleven(CHECK.getEntry(j)) == 1) {
 97                     HINT.add(i);
 98                     HINT.add(j);
 99                     return HINT;
100                 }
101             }
102         }
103         int jack = 0, queen = 0, king = 0;
104         for(int i = 1; i <= CHECK.getLength(); i++) {
105             if(CHECK.getEntry(i).getRankValue() == 11) {
106                 jack = i;
107             }
108             if(CHECK.getEntry(i).getRankValue() == 12) {
109                 queen = i;
110             }
111             if(CHECK.getEntry(i).getRankValue() == 13) {
112                 king = i;
113             }
114         }
115         if(jack > 0 && queen > 0 && king > 0) {
116             HINT.add(jack);
117             HINT.add(queen);
118             HINT.add(king);
119             return HINT;
120         }
121         return HINT;
122     }
123
124     protected void replaySteps() // Reads user input and increments through replay steps where
    possible
125     {
126         Scanner scan = new Scanner(System.in);
127         String enterkey = "Press 'Enter' key to increment through replay moves.";
128         while(!REPLAY.isEmpty()) { // While the replay queue is not empty
129             System.out.print(enterkey);
130             enterkey = scan.nextLine();
131             System.out.print(enterkey);
132             if(enterkey.equals("")) {
133                 System.out.println(REPLAY.dequeue());
134             }
135             if(REPLAY.getFront().charAt(0) == 'B') {
136                 System.out.println(REPLAY.dequeue());
137             }
138             while(!REPLAY.isEmpty() && REPLAY.getFront().charAt(0) == 'R') {
139                 System.out.println(REPLAY.dequeue());
140             }
141         }
142         System.out.println("\nNo moves left to replay.");
143     }
144 }
145
```