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

```
 1 import java.util.Arrays;
 2
 3 public class BoardTest extends Board
 4 {
 5     private final AListArray<Card> tstBRD = new AListArray<Card>(); //Test board
 6     private final AListArray<Card> tstDCK = new AListArray<Card>(); //Test deck
 7     private final Queue<String> REPLAY = new Queue<String>();
 8     private int counter;
 9
10     private void createWinTestBoard() // Creates 2 - 10 of Hearts in test Board
11     {
12         tstBRD.add(new Card(0,0));
13         tstBRD.add(new Card(1,0));
14         tstBRD.add(new Card(2,0));
15         tstBRD.add(new Card(3,0));
16         tstBRD.add(new Card(4,0));
17         tstBRD.add(new Card(5,0));
18         tstBRD.add(new Card(6,0));
19         tstBRD.add(new Card(7,0));
20         tstBRD.add(new Card(8,0));
21     }
22     private void createWinTestDeck() // Creates Jack - King of Hearts in test Deck
23     {
24         counter = 4;
25         tstDCK.add(new Card(9,0));
26         tstDCK.add(new Card(10,0));
27         tstDCK.add(new Card(11,0));
28         tstDCK.add(new Card(12,0));
29     }
30
31     private void createLoseTestBoard() // Creates 2 - 10 of Hearts in test Board
32     {
33         tstBRD.add(new Card(4,0));
34         tstBRD.add(new Card(5,0));
35         tstBRD.add(new Card(2,1));
36         tstBRD.add(new Card(6,3));
37         tstBRD.add(new Card(4,3));
38         tstBRD.add(new Card(4,1));
39         tstBRD.add(new Card(10,0));
40         tstBRD.add(new Card(3,3));
41         tstBRD.add(new Card(12,2));
42     }
43
44     private void createLoseTestDeck() // Creates 2 - 10 of Hearts in test Board
45     {
46         counter = 4;
47         tstDCK.add(new Card(7,3));
48         tstDCK.add(new Card(12,0));
49         tstDCK.add(new Card(9,2));
50         tstDCK.add(new Card(6,1));
51     }
52
53     public Card dealCard()  // deals a card from test deck
54     {
55         int length = counter;
56         Card deal = tstDCK.remove(length);
57         counter--;
58         return deal;
59     }
60
61     private void test()
62     {
63         System.out.println("\nCreating a testBoard and testDeck with only Cards that have suit 'Hearts
   ' ranging from Ace to King.");
64         createWinTestBoard();
65         createWinTestDeck();
66
67         System.out.println("\nLength of testBOARD card arrayList: " + tstBRD.getLength());
68         System.out.println("Length of testDECK card arrayList: " +tstDCK.getLength());
69
70         System.out.println("\nTest displayBoard method with full testBoard and remaining testDeck
   length.");
71         testDisplayBoard(tstBRD);
72
73         System.out.println("\nTest checkPossibleMoves method for possible moves and test replaceCards
   method if move is present");
```

```
 74         System.out.println("----Test on cards that should Win.----");
 75         while(tstBRD.getLength() > 0) {
 76             if (checkPossibleMoves(BOARD).isEmpty()) {
 77                 System.out.println("\nReturns slots that can be removed as integer ArrayList: " +
    checkPossibleMoves(tstBRD));
 78                 System.out.println("STALEMATE");
 79                 break;
 80                 }
 81             else if (checkPossibleMoves(tstBRD).getLength() > 2) {
 82                 System.out.println("\nReturns slots that can be removed as integer ArrayList: " +
    checkPossibleMoves(tstBRD));
 83                 System.out.println(tstBRD.toString());
 84                 testReplaceCards(checkPossibleMoves(tstBRD).getEntry(1), checkPossibleMoves(tstBRD).
    getEntry(2),
 85                         checkPossibleMoves(tstBRD).getEntry(3));
 86             }
 87             else {
 88                 System.out.println("\nReturns slots that can be removed as integer ArrayList: " +
    checkPossibleMoves(tstBRD));
 89                 System.out.println(tstBRD.toString());
 90                 testReplaceCards(checkPossibleMoves(tstBRD).getEntry(1), checkPossibleMoves(tstBRD).
    getEntry(2));
 91             }
 92         }
 93         System.out.println("\nBoard after test completed: " + tstBRD.toString());
 94
 95         System.out.println("\nTesting Replay functionality");
 96         while (!REPLAY.isEmpty()) {
 97             System.out.println(REPLAY.dequeue());
 98         }
 99
100         System.out.println("\nCreating a testBoard and testDeck with shuffled arrangement of cards
    that shouldn't be able to win");
101         createLoseTestBoard();
102         createLoseTestDeck();
103
104         System.out.println("\nTest checkPossibleMoves method for possible moves and test replaceCards
     method if move is present");
105         System.out.println("----Test on cards that should Lose.----");
106         while(tstBRD.getLength() > 0) {
107             if (checkPossibleMoves(tstBRD).isEmpty()) {
108                 System.out.println("\nReturns slots that can be removed as integer ArrayList: " +
    checkPossibleMoves(tstBRD));
109                 System.out.println(tstBRD.toString());
110                 System.out.println("STALEMATE");
111                 break;
112             }
113             else if (checkPossibleMoves(tstBRD).getLength() > 2) {
114                 System.out.println("\nReturns slots that can be removed as integer ArrayList: " +
    checkPossibleMoves(tstBRD));
115                 System.out.println(tstBRD.toString());
116                 testReplaceCards(checkPossibleMoves(tstBRD).getEntry(1), checkPossibleMoves(tstBRD).
    getEntry(2),
117                         checkPossibleMoves(tstBRD).getEntry(3));
118             }
119             else {
120                 System.out.println("\nReturns slots that can be removed as integer ArrayList: " +
    checkPossibleMoves(tstBRD));
121                 System.out.println(tstBRD.toString());
122                 testReplaceCards(checkPossibleMoves(tstBRD).getEntry(1), checkPossibleMoves(tstBRD).
    getEntry(2));
123             }
124         }
125
126         System.out.println("\nCurrent state of board after test");
127         testDisplayBoard(tstBRD);
128
129         try {
130             System.out.println("Test to replace slots 7 & 8 in tstBRD ArrayList.");
131             testReplaceCards(7, 8);
132         }
133         catch (IndexOutOfBoundsException e) {
134             System.out.println("IndexOutOfBoundsException error caught.");
135             System.out.println("8 is out of range.\n");
136         }
137         System.out.println("Changes made (if any): " + tstBRD.toString());
```

```
138
139         try {
140             System.out.println("\nTest to replace slots 0 and 1 in tstBRD ArrayList.");
141             testReplaceCards(0, 1);
142         }
143         catch (IndexOutOfBoundsException e) {
144             System.out.println("IndexOutOfBoundsException error caught.");
145             System.out.println("0 is out of range.\n");
146         }
147         System.out.println("Changes made (if any): " + tstBRD.toString());
148         System.out.println("\nIf changes made, must ensure accidental user selection is limited
    appropriately to prevent any " +
149                 "unwanted crashes or replacements.");
150     }
151
152
153     public static void main(String[] args)
154     {
155         BoardTest boardTest = new BoardTest();
156         boardTest.test();
157     }
158
159     //-----------------Test Methods adapted from Board Class using test variables--------------------
160
161
162     private void testReplaceCards(int replace1, int replace2)
163     {
164         REPLAY.enqueue("Board: " + tstBRD.toString());
165         int[] replace = {replace1,replace2};
166         Arrays.sort(replace);
167         for(int i = 1; i >= 0; i--) {
168             REPLAY.enqueue("Removed: "+ tstBRD.getEntry(replace[i]).toString());
169             if (tstDCK.getLength() > 0) {
170                 tstBRD.replace(replace[i], dealCard());
171             }
172             else {
173                 tstBRD.remove(replace[i]);
174             }
175         }
176     }
177
178     private void testReplaceCards(int replace1, int replace2, int replace3)
179     {
180         REPLAY.enqueue("Board: " + tstBRD.toString());
181         int[] replace = {replace1,replace2,replace3};
182         Arrays.sort(replace);
183         for(int i = 2; i >=0; i--) {
184             REPLAY.enqueue("Removed: "+ tstBRD.getEntry(replace[i]).toString());
185             if (tstDCK.getLength() > 0) {
186                 tstBRD.replace(replace[i], dealCard());
187             }
188             else {
189                 tstBRD.remove(replace[i]);
190             }
191         }
192     }
193     private void testDisplayBoard(AListArray<Card> CHECK)
194     {
195         System.out.println("\n-----------Board-------------");
196         for (int i = CHECK.getLength(); i > 0; i--)
197         {
198             System.out.println("Slot " + (i) + "  -  " + CHECK.getEntry(i));
199         }
200         System.out.println("----------------------------");
201         System.out.println(tstDCK.getLength() + " cards left in deck.");
202         System.out.println("----------------------------\n");
203     }
204 }
205
206
```