Testing - Elevens

Author

Mr. Adam Torok - B00798824

Jordanstown, November 26, 2021

Testing Strategy

Every basic part of the application is thoroughly tested. The strategy was to exclude the any error which is based on the data structures. This way we can only focus the logic part of the applications, dealing with extra calls or no calls at all. The fundamentals have to be right to make the debugging process easier, because on later stage, it is very hard to spot where is the error occurred. Below, we can see the `NodeTest.java`, in this file the pointer were tested and a link between them.

NodeTest.java

```
1  package arch;
2  import arch.*;
3  public class NodeTest {
4      public static void main(String[] args) {
5          Node<Integer> node1 = new Node<Integer>(1);
6          Node<Integer> node2 = new Node<Integer>(2);
7          Node<Integer> node3 = new Node<Integer>(3);
8          Node<Integer> node4 = new Node<Integer>(4);
9          Node<Integer> node5 = new Node<Integer>(5);
10
11         node1.setNext(node2);
12         node2.setNext(node3);
13         node3.setNext(node4);
14         node4.setNext(node5);
15         node5.setNext(null);
16         System.out.println("Testing The nodes pointing to each other");
17         System.out.println(node1.toString());
18
19         System.out.println("\n\nSetting All to null");
20         node1.setNext(null);
21         node2.setNext(null);
22         node3.setNext(null);
23         node4.setNext(null);
24         node5.setNext(null);
25         System.out.println(node1.toString());
26         System.out.println(node2.toString());
27         System.out.println(node3.toString());
28         System.out.println(node4.toString());
29         System.out.println(node5.toString());
30     }
31 }
```

Testing `Card` was a next step. The `Card` extends the `Node` class. The test is focused on to be able to only generate a valid card (based on value and suit).

CardTest.java

```java
package arch;

public class CardTest {
    public static void state(Card c) {
        System.out.println("Suit:\t\t\t" + c.getSuit());
        System.out.println("isFace:\t\t\t" + c.isFace());
        System.out.println("cardValue:\t\t" + c.getCardValue());
        System.out.println("callOut: \t\t" + c.toString() + "\n");
    }

    public static void main(String[] args) {

        //creation of an empty card
        System.out.println("\nCard empty = new Card();");
        try {
            Card empty = new Card();
        }catch(NullPointerException e){
            System.out.println(e);
        }


        // Creation of a card array
        LinkedList<String> myList = new LinkedList<>();
        System.out.println("\n\nCreating a set of suit:");
        Card[] cards = new Card[13];
        cards[0] = new Card(13, "S");
        cards[1] = new Card(12, "S");
        cards[2] = new Card(11, "S");
        cards[3] = new Card(10, "S");
        cards[4] = new Card(9, "S");
        cards[5] = new Card(8, "S");
        cards[6] = new Card(7, "S");
        cards[7] = new Card(6, "S");
        cards[8] = new Card(5, "S");
        cards[9] = new Card(4, "S");
        cards[10] = new Card(3, "S");
        cards[11] = new Card(2, "S");
        cards[12] = new Card(1, "S");

        for (Card c : cards) {
            state(c);
        }
        System.out.println("\n\nCreating an invalid suit:");
        try {
            Card card1 = new Card(1, "x");
        } catch (IllegalStateException ex) {
            System.out.println(ex.getMessage());
        }

        System.out.println("\n\nCreating w an invalid suit:");
        try {
            Card card1 = new Card(1, "x");
            Card card2 = new Card(15, "s");
        } catch (IllegalStateException ex) {
            System.out.println(ex.getMessage());
        }

        System.out.println("\n\nCreating card w invalid value:");
        try {
            Card card2 = new Card(15, "s");
        } catch (IllegalStateException ex) {
            System.out.println(ex.getMessage());
        }
    }
```

```
65 }
```

This is the generic linked list data type test. The tested parts were:

- insert an element
- remove an element from the middle
- remove an element from the first position
- remove an element from the last position

LinkedListTest.java

```
package arch;

public class LinkedListTest {

    public static void state(LinkedList list){
        try {
            System.out.println("Size:\t\t\t" + list.getSize());
            System.out.println("Empty:\t\t\t" + list.isEmpty());

            System.out.println("First Node:\t\t" + list.getFirstNode().getData());
            System.out.println("Last Node:\t\t" + list.getLastNode().getData());

            System.out.println("toString\t\t" + list.toString());
            System.out.println("toArray.length\t" + list.toArray().length);
        }catch(EmptyDeckException e){
            System.out.println("Empty List");

        }catch(NullPointerException e){
            System.out.println("Empty List");
        }
    }


    public static void main(String[] args) {
        // Creation of a linked List
        LinkedList<String> myList = new LinkedList<>();
        System.out.println("State Of the list:");
        state(myList);
        System.out.println("\n\nAdding a set of suit:");
        myList.addNewEntry("A");
        myList.addNewEntry("K");
        myList.addNewEntry("Q");
        myList.addNewEntry("J");
        myList.addNewEntry("T");
        myList.addNewEntry("9");
        myList.addNewEntry("8");
        myList.addNewEntry("7");
        myList.addNewEntry("6");
        myList.addNewEntry("5");
        myList.addNewEntry("4");
        myList.addNewEntry("3");
        myList.addNewEntry("2");
        state(myList);
            System.out.println("\n\nRemoving The First Element (2):");
            myList.removeFirstElement();
            state(myList);
            System.out.println("\n\nRemoving an element in the middle (T):");
            myList.removeSpfecific("T");
            state(myList);
            System.out.println("\n\nRemoving the last element (A):");
            System.out.println("The first element swaps slip to the deleted posish");
            myList.removeSpfecific("A");
            state(myList);
            System.out.println("\n\nEmpty the list:");
            myList.clear();
            state(myList);
        System.out.println("Remove an element from an empty list");
            try {
```

```
59                  myList.removeFirstElement();
60                  state(myList);
61              }catch (NullPointerException e ){
62                  System.out.println(e);
63              }
64
65      }
66  }
```

The following points were tested:

- creation of a deck
- creation of an empty deck (for testing)
- creation of a full but not shuffled deck (testing)
- creation of a full and shuffled deck
- visual test about a full deck (avoid, duplicated, they are in order etc.)
- the first and last pointers of the deck
- the deck ability to disable any addition or shuffle like operations (locked deck, add new card, double shuffle etc.)

DeckTest.java

```
1  package arch;
2
3  public class DeckTest {
4      public static void state(Deck d) {
5          System.out.println("Current Deck Size: " + d.getSize());
6          try {
7              System.out.println("First Card: " + d.getFirstCard().toString());
8              System.out.println("Last Card: " + d.getLastCard().toString());
9              System.out.println("Deck Size: " + d.toArray().length);
10             System.out.println("ToString: " + d.toString());
11             System.out.println("Shuffle: " + d.isLocked());
12             System.out.println("\n");
13         } catch (EmptyDeckException e) {
14             System.out.println("Empty Deck\n");
15         } catch (NullPointerException e) {
16             System.out.println("This card is not in the deck");
17         }
18     }
19
20     public static void shuffledState(Deck d) {
21         System.out.println("Current Deck Size: " + d.getSize());
22         try {
23             System.out.println("Shuffle: true");
24             System.out.println("First Card: " + d.getFirstCard().toString());
25             System.out.println("Last Card: " + d.getLastCard().toString());
26             System.out.println("Deck Size: " + d.toArray().length);
27             System.out.println("ToString: " + d.toString());
28             System.out.println("\n");
29         } catch (EmptyDeckException e) {
30             System.out.println("Empty Deck\n");
31         } catch (NullPointerException e) {
32             System.out.println("This card is not in the deck");
33         }
34     }
35
36     public static void main(String[] args) throws LockedDeckException, CardNotFoundException,
       EmptyDeckException {
37         try {
38             System.out.println("Crating and Empty Deck");
39             Deck dc = new Deck(false, false);
40             state(dc);
41             System.out.println("\nCreating A full but not shuffled deck");
42             Deck dd = new Deck(false, true);
43             state(dd);
44             System.out.println("Creating A full and shuffled deck");
45             Deck de = new Deck(true, true);
46             state(de);
47
48
49             System.out.println("DECK TEST:\n-------------");
50             Deck d = new Deck();
```

```
            state(d);
            System.out.println("REMOVE THE FIRST 4 K CARD:\n-------------");
            d.removeFirstElement();
            d.removeFirstElement();
            d.removeFirstElement();
            d.removeFirstElement();
            state(d);
            Card c = new Card(7, "s");
            System.out.println("Removing: " + c + "(connected two card [->][][<-] the first element doesnt change)\n-------------");
            d.removeFirstElement(c);
            state(d);
            Card c2 = new Card(12, "c");
            System.out.println("Removing first card: " + c2 + "\n-------------");
            d.removeFirstElement(c2);
            state(d);
            Card c3 = new Card(1, "h");
            System.out.println("Removing last card: " + c3 + "\n-------------");
            d.removeFirstElement(c3);
            state(d);
            Card c4 = new Card(1, "d");
            System.out.println("Removing last card: " + c4 + "\n-------------");
            d.removeFirstElement(c4);
            state(d);
            System.out.println("Creation of an empty deck:\n-------------");
            Deck emptyDeck = new Deck(false, false);
            state(emptyDeck);
            System.out.println("\nAdd just a couple of card to the deck:\n-------------");
            emptyDeck.addNewEntry(c);
            emptyDeck.addNewEntry(c2);
            emptyDeck.addNewEntry(c3);
            emptyDeck.addNewEntry(c4);
            state(emptyDeck);
            Card c5 = new Card(12, "h");
            System.out.println("Remove a non-existing card from the deck: " + c5 + "\n-------------");
            emptyDeck.removeFirstElement(c5);
            state(emptyDeck);
        } catch (CardNotFoundException e) {
            System.out.println("Card not found: " + e + "\n");
        }

        System.out.println("Testing shuffle twice\n-----------------");
        try {
            Deck sd = new Deck(true, true);
            sd.shuffle();
            shuffledState(sd);

        } catch (LockedDeckException e) {
            System.out.println(e);
        }
    }
}
```

The most complicated test:

- board size
- KJQ numbers
- first and last pointers
- sorting
- max and minimum sizes
- validate user input
- removal handling
- separate card value returns

BoardTest.java

```
package arch;
public class BoardTest {

    public static void boardState(Board b){
        System.out.println("-------------------------");
        System.out.println("\nCurrent Board Size: " + b.getSize());
            System.out.println("First Card: " + b.getFirstCard().toString());
            System.out.println("Last Card: " + b.getLastCard().toString());
            System.out.println("Number of J's: " + b.getcJ());
            System.out.println("Number of Q's: " + b.getcQ());
            System.out.println("Number of K's: " + b.getcK());
            try {
                System.out.println("Board Size: " + b.toArray().length);
                System.out.println("Board: " + b);
                System.out.println("\n");
            }catch (NullPointerException e){
                System.out.println("The board is empty");
            }
        System.out.println("-------------------------");
    }
    public static void state(Deck d){
        System.out.println("Current Deck Size: " + d.getSize());
        try {
            System.out.println("First Card: " + d.getFirstCard().toString());
            System.out.println("Last Card: " + d.getLastCard().toString());
            System.out.println("Deck Size: " + d.toArray().length);
            System.out.println("ToString: " + d);
            System.out.println("Shuffle: false");
            System.out.println("\n");
        }catch(EmptyDeckException e){
            System.out.println("Empty Deck\n");
        }catch (NullPointerException e){
            System.out.println("This card is not in the deck");
        }
    }
    public static void states(Deck d, Board b){
        state(d);
        boardState(b);
        state(d);
    }

    public static void main(String[] args) throws LockedDeckException, EmptyDeckException {
        Deck d = new Deck();
        Board b = new Board();
        System.out.println("Fill up the board with the first 9 cards\n");
        d.shuffle();
        for (int i = 0; i <= 8; i++) {
            b.addNewEntry(d.removeFirstElement());
        }
        states(d,b);
```

```
51
52
53          System.out.println("Build a static board\n");
54
55          Board ba = new Board();
56
57          Card c1 = new Card(1,"h");
58          Card c2 = new Card(2,"h");
59          Card c3 = new Card(3,"h");
60          Card c4 = new Card(4,"h");
61          Card c5 = new Card(5,"h");
62          Card c6 = new Card(6,"h");
63          Card c7 = new Card(7,"h");
64          Card c8 = new Card(8,"h");
65          Card c9 = new Card(9,"h");
66          ba.addNewEntry(c1);
67          ba.addNewEntry(c2);
68          ba.addNewEntry(c3);
69          ba.addNewEntry(c4);
70          ba.addNewEntry(c5);
71          ba.addNewEntry(c6);
72          ba.addNewEntry(c7);
73          ba.addNewEntry(c8);
74          ba.addNewEntry(c9);
75          boardState(ba);
76          try {
77              System.out.println("Remove card: " + c3);
78              ba.removeACard(c3);
79              System.out.println("Get Card Value of the first element: " + ba.getNthCardValue(1));
80              System.out.println("Get Card Value of the 2 element: " + ba.getNthCardValue(2));
81              System.out.println("Get Card Value of the 3 element: " + ba.getNthCardValue(3));
82              System.out.println("Get Card Value of the 4 element: " + ba.getNthCardValue(4));
83              System.out.println("Get Card Value of the 6 element: " + ba.getNthCardValue(5));
84              System.out.println("Get Card Value of the 7 element: " + ba.getNthCardValue(6));
85              System.out.println("Get Card Value of the 8 element: " + ba.getNthCardValue(7));
86              System.out.println("Get Card Value of the last element: " + ba.getNthCardValue(8));
87          }catch(CardNotFoundException e){
88              System.out.println("Card not in the board");
89          }
90          boardState(ba);
91
92          try {
93              System.out.println("Remove a non-existing Card: " + new Card(10,"s"));
94              ba.removeACard(new Card(10,"s"));
95          }catch(CardNotFoundException e){
96              System.out.println("Card not in the board");
97          }
98          boardState(ba);
99
100
101         try {
102             System.out.println("Remove first Card: " + new Card(1,"h"));
103             ba.removeACard(new Card(1,"h"));
104         }catch(CardNotFoundException e){
105             System.out.println("Card not in the board");
106         }
107         boardState(ba);
108
109         try {
110             System.out.println("Remove last Card: " + new Card(9,"h"));
111             ba.removeACard(new Card(9,"h"));
112         }catch(CardNotFoundException e){
113             System.out.println("Card not in the board");
114         }
115         boardState(ba);
116
117         try {
118             System.out.println("Remove nth: 3 (4 of hearts)");
119             ba.removeNthCard(3);
120         }catch(NullPointerException e){
```

```
            System.out.println("Card not in the board");
        }
        boardState(ba);

        try {
            System.out.println("Remove the first element: 1");
            ba.removeNthCard(1);
        }catch(NullPointerException e){
            System.out.println("Card not in the board");
        }
        boardState(ba);

        try {
            System.out.println("Remove the last element: 4");
            ba.removeNthCard(4);
        }catch(NullPointerException e){
            System.out.println("Card not in the board");
        }
        boardState(ba);
        ba.clear();
        Card c10 = new Card(11,"h");
        Card c11 = new Card(12,"c");
        Card c12 = new Card(13,"s");
        ba.addNewEntry(c1);
        ba.addNewEntry(c2);
        ba.addNewEntry(c3);
        ba.addNewEntry(c4);
        ba.addNewEntry(c5);
        ba.addNewEntry(c6);
        ba.addNewEntry(c10);
        ba.addNewEntry(c11);
        ba.addNewEntry(c12);
        boardState(ba);
        try {
            System.out.println("Testing plain 11's (true)");
            System.out.println(ba.checkAnswer(2, 3, 0));
            System.out.println(ba.checkAnswer(1, 4, 0));
            System.out.println(ba.checkAnswer(1, 4, 0));
            System.out.println("Testing plain 11's (false)");
            System.out.println(ba.checkAnswer(7, 8, 0));
            System.out.println(ba.checkAnswer(7, 9, 0));
            System.out.println("Testing plain 11's (KJQ)");
            System.out.println(ba.checkAnswer(7, 8, 9));

            //System.out.println("Testing Fail Inputs"); //uncomment to test
            //System.out.println(ba.checkAnswer(0, 0, 0));
            //System.out.println(ba.checkAnswer(1, 0, 0));
            //System.out.println(ba.checkAnswer(0, 1, 0));
            //System.out.println(ba.checkAnswer(1, 1, 1));
            //System.out.println(ba.checkAnswer(1, 1, 1));
            //System.out.println(ba.checkAnswer(1, 1, 0));
            //System.out.println(ba.checkAnswer(0, 1, 1));
        }catch(NullPointerException e){
            System.out.println(e);
        }

        System.out.println("Testing possible valid moves");
        Deck db = new Deck();
        Board bb = new Board();
        System.out.println("Fill up the board with the first 9 cards\n");
        db.shuffle();
        for (int i = 0; i <= 8; i++) {
            bb.addNewEntry(d.removeFirstElement());
        }
        boardState(bb);



        Board bc = new Board();
        bc.addNewEntry(new Card(6,"c"));
```

```
        bc.addNewEntry(new Card(5,"d"));


        boardState(bc);
        bc.searchValidMoves();
        Stack st = bc.getValidMove();
        System.out.println("Valid Moves: " + st.getSize());
        bc.removeNthCard(1);
        bc.removeNthCard(1);
        bc.addNewEntry(new Card(11,"c"));
        bc.addNewEntry(new Card(12,"d"));
        bc.addNewEntry(new Card(13,"d"));
        boardState(bc);
        bc.searchValidMoves();
        st = bc.getValidMove();
        System.out.println("Valid Moves: " + st.getSize());


        Board be = new Board();
        be.addNewEntry(new Card(1,"s"));
        be.addNewEntry(new Card(1,"d"));
        be.addNewEntry(new Card(5,"c"));
        be.addNewEntry(new Card(6,"s"));
        be.addNewEntry(new Card(7,"d"));
        be.addNewEntry(new Card(9,"h"));
        be.addNewEntry(new Card(9,"d"));
        be.addNewEntry(new Card(13,"c"));
        be.addNewEntry(new Card(13,"h"));
      boardState(be);

      be.searchValidMoves();
      System.out.println(be.getValidMove().getSize());


    }
}
```

Testing were include:

- push and pop operations
- stack item stored in reverse order
- peek method
- popping on an empty stack

StackTest.java

```
package arch;

public class StackTest {


    public static void main(String[] args) {
        Stack s = new Stack();
        int[] p1 = {3, 6, 9};
        int[] p2 = {0, 1, 1};
        int[] p3 = {2, 9, 0};
        int[] p4 = {1, 0, 1};

        System.out.println("Pushing {3, 6, 9}  - order is always " +
                "descending in a stack element");
        s.push(p1);
        System.out.println(s);
        System.out.println("Size of the stack is: " + s.getSize());

        System.out.println("Pushing {0,1,1}");
        s.push(p2);
        System.out.println(s);
        System.out.println("Size of the stack is: " + s.getSize());
        System.out.println("Pushing {2, 9, 0}");
        s.push(p3);
        System.out.println(s);
        System.out.println("Size of the stack is: " + s.getSize());
        System.out.println("Pushing {1, 0, 1}");
        s.push(p4);
        System.out.println(s);
        System.out.println("Size of the stack is: " + s.getSize());
        System.out.println("Content of the Stack");
        System.out.println(s);
        System.out.println("Popping one item");
        s.pop();
        System.out.println(s);
        System.out.println("Testing Peak");
        int[] peek = s.peek();
        for (int i : peek){
            System.out.println(i);
        }
        System.out.println(s);
        System.out.println("Popping the three remaining element");
        s.pop();
        s.pop();
        s.pop();
        System.out.println(s);
        System.out.println("Try to pop with no elements at all (null return)");
        System.out.println(s.pop());
    }
}
```

Demo test is used to quickly start the application id demonstration mode and see the result.

DemoTest.java

```
package arch;

public class DemoTest {
```

```
4
5       public static void testTemodnstrationMode() throws LockedDeckException, EmptyDeckException {
6           App app = new App();
7           Deck d = new Deck(true,true);
8           Board b = new Board();
9           app.playMode(true, true);
10      }
11
12      public static void main(String[] args) throws LockedDeckException, EmptyDeckException {
13          testTemodnstrationMode();
14
15      }
16  }
```

Manual test is for simply test the application in manual mode (including the menu).

ManualTest.java

```
1   package arch;
2
3   public class ManualTest {
4
5       public static void main(String[] args) throws LockedDeckException, EmptyDeckException {
6           App app = new App();
7           app.selectMode();
8
9       }
10  }
```

Custom Exceptions

EmptyBoardException.java

```
package arch;

public class EmptyBoardException extends Exception{
    public  EmptyBoardException(Card c) {
        super(c.toString());
    }
}
```

EmptyDeckException.java

```
package arch;
import arch.*;
public class EmptyDeckException extends Exception{
    public EmptyDeckException(String s) {
        super(s);
    }
}
```

LockedDeckException.java

```
package arch;
import arch.*;
public class LockedDeckException extends Exception{
    public LockedDeckException(String s) {
        super(s);
    }
}
```

CardNotFoundException.java

```
package arch;

public class CardNotFoundException extends Exception{
    public  CardNotFoundException(Card c) {
        super(c.toString());
    }
}
```

This is pdfTeX, Version 3.141592653-2.6-1.40.22 (TeX Live 2021/Arch Linux) kpathsea version 6.3.3