

Table of Contents

Introduction	5
Task -1 (Application)	6
Overview of all Project Files	6
Main package files	6
Test package files	7
Class: NotationReader	8
Class: Dice	10
Class: HistoryDice	13
Class: Card	14
Class: CardStack	16
Class: CardStackRemovedCards	18
Class: CollectionCardStacks	19
Class: CardStackDealtCards	22
Class: UserInterface	23
Inner Class: MyButton	29
Interface: NotificationListener	29
Interface: DrawActionListener	30
Class: Canvas	31
Task-2 (Testing Data)	37
Test Suite No.1	37
Test Execution	37
Test Result	41
Test Summary	41
Test Suite No.2	42
Test Execution	43
Test Summary	44
Test Suite No.3	45
Test Execution	45
Test Result	49
Test Summary	49
Test Suite No.4	50
Test Execution	50
Test Summary	54

Test Suite No.5	55
Test Execution	55
Test Result	57
Test Summary	57
Test Suite No.6	58
Test Execution	58
Test Summary	59
Test Suite No.7	60
Test Execution	60
Test Result	64
Test Summary	64
Test Suite No.8	65
Test Execution	65
Test Summary	68
Test Suite No.9	69
Test Execution	69
Test Result	71
Test Summary	71
Test Suite No.10	72
Test Execution	72
Test Summary	74
Test Suite No. 11	75
Test Execution	75
Test Result	77
Test Summary	77
Test Suite No.12	78
Test Execution	78
Test Summary	79
Test Suite No. 13	80
Test Execution	80
Test Result	81
Test Summary	81
Test Suite No. 14	82
Test Execution	82
Test Summary	83

Test Suite No.15	84
Test Execution	84
Test Result	86
Test Summary	86
Test Suite No.16	87
Test Execution	87
Test Summary	90
Test Suite No.17	91
Test Execution	91
Test Result	94
Test Summary	94
Test Suite No.18	95
Test Execution	95
Test Summary	97
Test Suite No.19	98
Test Execution	98
Test Result	100
Test Summary	100
Test Suite No.20	101
Test Execution	101
Test Summary	103
Test Suite No.21	104
Test Execution	104
Test Result	109
Test Summary	109
Test Suite No.22	110
Test Execution	110
Test Result	116
Test Summary	116
Test Suite No.23	117
Test Execution	117
Test Result	122
Test Summary	122
Test Suite No.24	123
Test Execution	123

Test Result	128
Test Summary	128
Final Test Suite No.25.....	129
Test Execution.....	129
Test Summary	135
Task 3 – Class Diagram.....	136
Conclusion	137
Referencing	138

Introduction

In games that use the element of randomness to the gameplay, often times, regular faced dices are used. However, for the dice to produce an ideal distribution, the time taken is relatively longer than the time period of game itself. One of the ways of mitigating this situation, is the use of card stack, where the card stack stores the card that corresponds to the combination generated by the dice roll, if in case the dice was rolled.

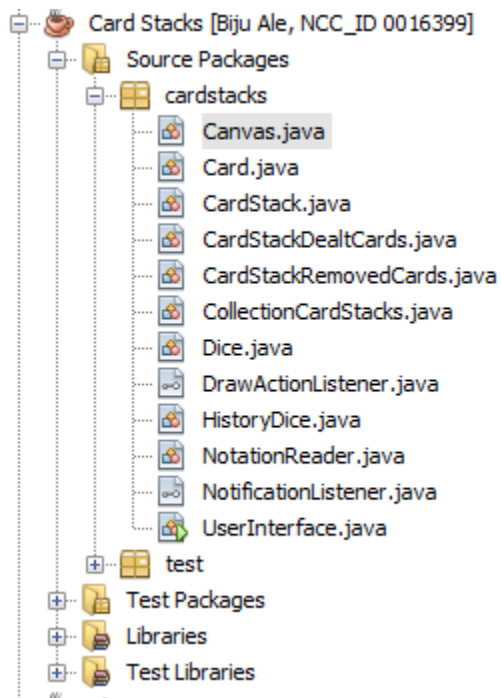
The java based application presented in this documentation, is specifically, and designed to solve the issue with regular dice rolls. In addition to storing the combination in the form of a card, it also promotes a fair randomness by optimizing the occurrence of each combination towards statistically favorable distribution. Moreover, it has many advantages over real life card stack adaptation. It can generate virtually any kind of dices which means it is able to simulate the rolling of multi-dimensional dices that is not possible in real world. It can independently track multiple type of dice's rolling's& maintain graphs of frequencies

This report incorporates mainly 3 parts – the source code of the application, the testing carried out full coverage of verification & validation of all classes, and finally the UML based class diagram to provide the overview of the application's design.

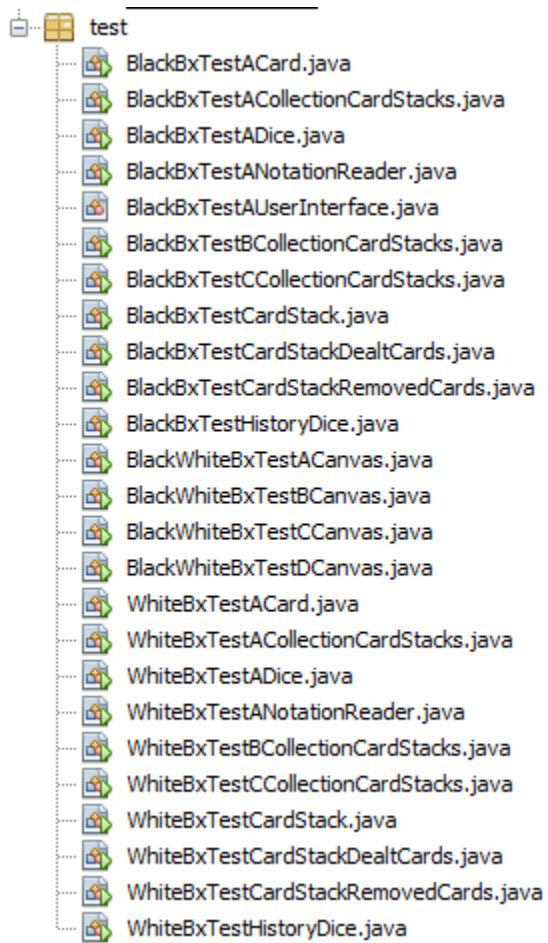
Task -1 (Application)

Overview of all Project Files

Main package files



Test package files



Class: NotationReader

[PLEASE TURN OVER]


```

6   package cardstacks;
7
8   import java.util.regex.Pattern;
9
10  /**
11   *
12   * @author Biju Ale
13   */
14  public class NotationReader {
15
16      private int numDices, numFaces, toRemove;
17      private String diceNotation;
18
19      public int getNumDices() {
20          return numDices;
21      }
22
23      public int getNumFaces() {
24          return numFaces;
25      }
26
27      public int getToRemove() {
28          return toRemove;
29      }
30
31      public String getDiceNotation() {
32          return diceNotation;
33      }
34
35      public void parseDiceNotation(String diceNotation) throws Exception {
36          String[] parts;
37          //Main part in Regex Pattern: (1|[1-9][0-9]*) [Matches any number without leading↵
38          //zeros.]
39          if (Pattern.matches("((1|[1-9][0-9]*)d(1|[1-9][0-9]*))|((1|[1-9][0-9]*)d(1|[1-9][↵
40          0-9]*)[-](1|[1-9][0-9]*))", diceNotation)) {
41              parts = diceNotation.split("d|-");
42              this.diceNotation = diceNotation;
43              this.numDices = Integer.parseInt(parts[0]);
44              this.numFaces = Integer.parseInt(parts[1]);
45              this.toRemove = (diceNotation.contains("-")) ? Integer.parseInt(parts[2]) : 0↵
46          ; //Ternary Operator
47          } else {
48              throw new Exception("\n\nINVALID DICE NOTATION!");
49          }
50      }
51  }

```

Class: Dice

[PLEASE TURN OVER]

```

6  package cardstacks;
7
8  import java.util.Arrays;
9  import java.util.Random;
10
11  /**
12   *
13   * @author Biju Ale
14   */
15  public class Dice {
16
17      private String diceName;
18      public static final int ROLL_TIMES = 10000;
19      private Integer[] Combinations;
20      private Integer[] frequencies;
21      private int minCombination, maxCombination;
22
23      public Integer[] getCombinations() {
24          return Combinations;
25      }
26
27      public Integer[] getFrequencies() {
28          return frequencies;
29      }
30
31      public String getDiceName() {
32          return diceName;
33      }
34
35      public int getMinCombination() {
36          return minCombination;
37      }
38
39      public int getMaxCombination() {
40          return maxCombination;
41      }
42
43      public Dice(NotationReader nreader) throws Exception {
44          if (nreader.getToRemove() > (nreader.getNumDices() * nreader.getNumFaces())) {
45              throw new Exception("No. of cards to remove cannot exceed total no. of cards.↵
46              Enter valid notation.\n");
47          }
48      }
49  }

```

```

47         this.diceName = nreader.getDiceNotation();
48         setMinMax(nreader.getNumDices(), nreader.getNumDices() * nreader.getNumFaces());
49     }
50
51     //set - min & max Combination
52     private void setMinMax(int minCombination, int maxCombination) {
53         this.minCombination = minCombination;
54         this.maxCombination = maxCombination;
55         populateCombinations();
56     }
57
58     //Populate all possible combinations in Combination array
59     private void populateCombinations() {
60         Combinations = new Integer[maxCombination - minCombination + 1];
61         int index = 0;
62         for (int eachCombination = minCombination; eachCombination < maxCombination + 1; ↵
eachCombination++) {
63             Combinations[index] = eachCombination;
64             index++;
65         }
66         roll(Combinations);
67     }
68
69     //Roll the dice & record combinations' frequencies
70     private void roll(Integer[] Combinations) {
71         int randomIndex;
72
73         frequencies = new Integer[Combinations.length];
74
75         Arrays.fill(frequencies, 0); //Reset all indexes
76         //Save index frequencies
77         Random rdmGenerator = new Random();
78         for (int i = 0; i < ROLL_TIMES; i++) {
79             randomIndex = rdmGenerator.nextInt(Combinations.length);
80             frequencies[randomIndex] += 1;
81         }
82     }
83 }
84

```

Class: HistoryDice

```
6      package cardstacks;
7
8      import java.util.ArrayList;
9
10     /**
11     *
12     * @author Biju Ale
13     */
14     public class HistoryDice extends ArrayList<Dice> {
15
16     public boolean addToDiceHistory(Dice dice) {
17         for (Dice eachDice : this) {
18             if (eachDice.getDiceName().equals(dice.getDiceName())) {
19                 return false;
20             }
21         }
22         return add(dice);
23     }
24 }
25
```

Class: Card

[PLEASE TURN OVER]

```

6  package cardstacks;
7
8  /**
9   *
10  * @author Biju Ale
11  */
12  public class Card {
13
14      private String diceNotation;
15      private int number;
16      private int frequency;
17
18      public Card(int number, int frequency, String diceNotation) {
19          this.number = number;
20          this.frequency = frequency;
21          this.diceNotation = diceNotation;
22      }
23
24      Card() {
25      }
26
27      public String getDiceNotation() {
28          return diceNotation;
29      }
30
31      public void setDiceNotation(String diceNotation) {
32          this.diceNotation = diceNotation;
33      }
34
35      public int getNumber() {
36          return number;
37      }
38
39      public void setNumber(int number) {
40          this.number = number;
41      }
42
43      public int getFrequency() {
44          return frequency;
45      }
46
47      public void setFrequency(int frequency) {
48          this.frequency = frequency;
49      }
50  }
51

```

Class: CardStack

[PLEASE TURN OVER]


```

6   package cardstacks;
7
8   import java.util.Collections;
9   import java.util.Iterator;
10  import java.util.LinkedList;
11
12  /**
13   *
14   * @author Biju Ale
15   */
16  public class CardStack extends LinkedList<Card> {
17
18      private String diceNotation;
19      private Integer max;
20
21      public String getDiceNotation() {
22          return diceNotation;
23      }
24
25      public CardStack(Dice dice, NotationReader nreader, CardStackRemovedCards csrc) {
26          this.diceNotation = dice.getDiceName();
27          populateCardStack(dice.getCombinations(), dice.getFrequencies());
28          if (nreader.getToRemove() > 0) {
29              removeCard(nreader.getToRemove(), csrc);
30          }
31      }
32
33      public CardStack(String diceNotation) {
34          this.diceNotation = diceNotation;
35      }
36
37      private void populateCardStack(Integer[] Combinations, Integer[] frequencies) {
38          for (int i = 0; i < Combinations.length; i++) {
39              add(new Card(Combinations[i], frequencies[i], diceNotation));
40          }
41          Collections.shuffle(this);
42      }
43
44      private void removeCard(int toRemove, CardStackRemovedCards csrc) {
45          Iterator itr = this.iterator();
46          for (int i = 0; i < toRemove; i++) {
47              csrc.add(getFirst());
48              removeFirst();
49          }
50      }
51
52  }
53

```

Class: CardStackRemovedCards

```
6   package cardstacks;
7
8   import java.util.ArrayList;
9
10  /**
11   *
12   * @author Biju Ale
13   */
14  public class CardStackRemovedCards extends ArrayList<Card> {
15
16      public ArrayList<Card> getRemovedCards(String diceNotation) {
17          ArrayList<Card> removedCards = new ArrayList();
18          for (Card eachCard : this) {
19              if (eachCard.getDiceNotation().equals(diceNotation)) {
20                  removedCards.add(eachCard);
21              }
22          }
23          return removedCards;
24      }
25  }
26
```

Class: CollectionCardStacks

[PLEASE TURN OVER]

```

6 package cardstacks;
7
8 import java.util.ArrayList;
9 import java.util.Collections;
10 import java.util.LinkedList;
11 import java.util.ListIterator;
12
13 /**
14  *
15  * @author Biju Ale
16  */
17 public class CollectionCardStacks extends LinkedList<CardStack> {
18
19     private NotificationListener notificationListener;
20
21     public Card moveDealtCard(String diceNotation, CardStackDealtCards csdc) {
22         Card dealtCard = new Card();
23         for (CardStack eachCardStack : this) {
24             if (eachCardStack.getDiceNotation().equals(diceNotation)) {
25                 if (!eachCardStack.isEmpty()) {
26                     dealtCard = eachCardStack.getFirst();
27                     eachCardStack.removeFirst();
28                     csdc.add(dealtCard);
29                 } else if (eachCardStack.isEmpty()) {
30                     notificationListener.send("\n\nCardStack empty! Reshuffling now...");
31                     rePopulateStack(diceNotation, csdc);
32                     dealtCard = eachCardStack.getFirst();
33                     eachCardStack.removeFirst();
34                     csdc.add(dealtCard);
35                 }
36             }
37         }
38         return dealtCard;
39     }
40
41     private void rePopulateStack(String diceNotation, CardStackDealtCards csdc) {
42         ListIterator<Card> itr = csdc.listIterator();
43         ArrayList<Card> toReturn = new ArrayList<Card>();
44         while (itr.hasNext()) {
45             Card next = itr.next();
46             if (next.getDiceNotation().equals(diceNotation)) {
47                 toReturn.add(next);

```

```

48         itr.remove();
49     }
50 }
51
52 for (CardStack eachCardStack : this) {
53     if (eachCardStack.getDiceNotation().equals(diceNotation)) {
54         eachCardStack.addAll(toReturn);
55     }
56 }
57
58 public boolean shuffleStack(String diceNotation) {
59     for (CardStack eachCardStack : this) {
60         if (eachCardStack.getDiceNotation().equals(diceNotation)) {
61             Collections.shuffle(eachCardStack);
62             return true;
63         }
64     }
65     return false;
66 }
67
68 public CardStack getFutureCards(String diceNotation) {
69     for (CardStack eachCardStack : this) {
70         if (eachCardStack.getDiceNotation().equals(diceNotation)) {
71             return eachCardStack;
72         }
73     }
74     return null;
75 }
76
77 public void addNotificationListener(NotificationListener notificationListener) {
78     this.notificationListener = notificationListener;
79 }
80 }
81

```

Class: CardStackDealtCards

```
6  package cardstacks;
7
8  import java.util.ArrayList;
9
10 /**
11  *
12  * @author Biju Ale
13  */
14 public class CardStackDealtCards extends ArrayList<Card> {
15
16     public ArrayList<Card> getDealtCards(String diceNotation) {
17
18         ArrayList<Card> dealtCards = new ArrayList();
19         for (Card eachCard : this) {
20             if (eachCard.getDiceNotation().equals(diceNotation)) {
21                 dealtCards.add(eachCard);
22             }
23         }
24         return dealtCards;
25     }
26 }
27
```

Class: UserInterface

[PLEASE TURN OVER]

```

6     package cardstacks;
7
8     import java.awt.BorderLayout;
9     import java.awt.Color;
10    import java.awt.Container;
11    import java.awt.Dimension;
12    import java.awt.FlowLayout;
13    import java.awt.Font;
14    import java.awt.Graphics;
15    import java.awt.GridLayout;
16    import java.awt.event.ActionEvent;
17    import java.awt.event.ActionListener;
18    import javax.swing.BorderFactory;
19    import javax.swing.JButton;
20    import javax.swing.JFrame;
21    import javax.swing.JLabel;
22    import javax.swing.JPanel;
23    import javax.swing.JScrollPane;
24    import javax.swing.JTextArea;
25    import javax.swing.JTextField;
26    import javax.swing.border.BevelBorder;
27    import javax.swing.border.EtchedBorder;
28
29    /**
30     *
31     * @author Biju Ale
32     */
33    public class UserInterface extends JFrame implements ActionListener, ↵
34    NotificationListener {
35
36        private JLabel lblEnterNotation;
37        private JTextField txtDiceNotation;
38        private static JTextArea txtNotification;
39        private JButton btnSubmit, btnFutureCards, btnRemoved, btnDealt, btnShuffle, ↵
40        btnGetCard;
41        private JPanel pnlUserInput, pnlCommands, pnlNotification;
42
43        private NotationReader nreader;
44        private HistoryDice historyDice;
45        private CardStack cardStack;
46        private CollectionCardStacks collectionCardStacks;
47        private CardStackRemovedCards cardStackRemovedCards;

```



```

47 private CardStackDealtCards cardStackDealtCards;
48 private Graphics g;
49 private static DrawActionListener drawActionListener;
50 private static Canvas canvas;
51
52
53 //Setting up Frame Properties
54 setSize(800, 850);
55 setTitle("Card Stacks - authored by Biju Ale");
56 setDefaultCloseOperation(EXIT_ON_CLOSE);
57
58 //Stting up Container
59 Container c = getContentPane();
60 c.setLayout(new BorderLayout());
61 c.setBackground(Color.decode("#F7E3CB"));
62
63 //Instantiating components at NORTH (for input)
64 lblEnterNotation = new JLabel("Enter dice notation");
65 lblEnterNotation.setFont(new Font("Times New Roman", Font.BOLD, 20));
66
67 txtDiceNotation = new JTextField(8);
68 txtDiceNotation.setBorder(BorderFactory.createCompoundBorder(BorderFactory.createEtchedBorder(EtchedBorder.RAISED, Color.GRAY, Color.BLUE), BorderFactory.createEmptyBorder(5, 5, 5, 5)));
69 txtDiceNotation.setPreferredSize(new Dimension(30, 30));
70 txtDiceNotation.setFont(new Font("Times New Roman", Font.BOLD, 20));
71
72 btnSubmit = new JButton("SUBMIT");
73 btnSubmit.setFont(new Font("Times New Roman", Font.BOLD, 20));
74 btnSubmit.setBackground(Color.DARK_GRAY);
75 btnSubmit.setForeground(Color.LIGHT_GRAY);
76
77 //Instantiating components at WEST (for commands)
78 btnFutureCards = new MyButton("PEEK FUTURE CARDS");
79 btnRemoved = new MyButton("PEEK REMOVED CARD");
80 btnDealt = new MyButton("PEEK DEALT CARDS");
81 btnShuffle = new MyButton("SHUFFLE");
82 btnGetCard = new MyButton("GET CARD");
83
84 //Instantiating panels for input, commands & notifciation
85 pnlUserInput = new JPanel(new FlowLayout(FlowLayout.CENTER));

```

```

86 pnlUserInput.setBackground(Color.decode("#1DE9B6"));
87 pnlCommands = new JPanel(new GridLayout(5, 1));
88 pnlNotification = new JPanel(new GridLayout(1, 1));
89
90 //Adding all components to input panel
91 pnlUserInput.add(lblEnterNotation);
92 pnlUserInput.add(txtDiceNotation);
93 pnlUserInput.add(btnSubmit);
94
95 //Adding all components to commands panel
96 pnlCommands.add(btnGetCard);
97 pnlCommands.add(btnShuffle);
98 pnlCommands.add(btnDealt);
99 pnlCommands.add(btnRemoved);
100 pnlCommands.add(btnFutureCards);
101
102 //Adding notification area
103 txtNotification = new JTextArea(3, 50);
104 txtNotification.setFont(new Font("Times New Roman", Font.PLAIN, 20));
105 txtNotification.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));
106 txtNotification.setBackground(Color.decode("#1DE9B6"));
107 JScrollPane txtNotificationScroll = new JScrollPane(txtNotification);
108
109 c.add(pnlUserInput, BorderLayout.PAGE_START);
110 c.add(pnlCommands, BorderLayout.LINE_START);
111 c.add(txtNotificationScroll, BorderLayout.PAGE_END);
112
113 //Adding action listener to all buttons
114  btnSubmit.addActionListener(this);
115  btnShuffle.addActionListener(this);
116  btnRemoved.addActionListener(this);
117  btnFutureCards.addActionListener(this);
118  btnGetCard.addActionListener(this);
119  btnDealt.addActionListener(this);
120
121 //Initialize all stacks, dice & cardstack history
122 historyDice = new HistoryDice();
123 cardStackRemovedCards = new CardStackRemovedCards();
124 cardStackDealtCards = new CardStackDealtCards();
125 collectionCardStacks = new CollectionCardStacks();
126  collectionCardStacks.addNotificationListener(this);
127

```

```

128 //Instantiating canvas
129 canvas = new Canvas();
130 canvas.addNotificationListener(this);
131 canvas.setPreferredSize(new Dimension(6000, 2000));
132 canvas.setBackground(Color.decode("#F7E3CB"));
133 JScrollPane canvasScroll = new JScrollPane(canvas);
134 c.add(canvasScroll, BorderLayout.CENTER);
135
136 drawActionListener = canvas;
137 setLocationRelativeTo(null);
138 }
139
140 @Override
141 public void actionPerformed(ActionEvent e) {
142     JButton btnSrc = (JButton) e.getSource();
143     if (btnSrc.equals(btnSubmit)) {
144         String diceNotation = txtDiceNotation.getText();
145         try {
146             nreader = new NotationReader();
147             nreader.parseDiceNotation(diceNotation);
148             Dice dice = new Dice(nreader);
149             if (historyDice.addToDiceHistory(dice)) {
150                 cardStack = new CardStack(dice, nreader, cardStackRemovedCards);
151                 collectionCardStacks.add(cardStack);
152                 txtNotification.append("\n\nRolling...\n" + diceNotation + "'s ↵
combinations & frequencies added to card stack.");
153                 btnGetCard.doClick();
154             } else {
155                 txtNotification.append("\n\nAlready rolled dice - " + diceNotation);
156                 btnGetCard.doClick();
157             }
158         } catch (Exception ex) {
159             txtNotification.append(ex.getMessage());
160             txtDiceNotation.requestFocus();
161         }
162     } else if (btnSrc.equals(btnGetCard)) {
163         canvas.sendSingleDealtCard(collectionCardStacks.moveDealtCard(↵
txtDiceNotation.getText(), cardStackDealtCards));
164         canvas.ACTION_DRAW = Canvas.DRAW_FOR_GET_CARD;
165     } else if (btnSrc.equals(btnDealt)) {
166         canvas.sendAllDealtCards(cardStackDealtCards.getDealtCards(txtDiceNotation.↵

```

```

167         canvas.ACTION_DRAW = Canvas.DRAW_FOR_DEALT_CARD;
168     } else if (btnSrc.equals(btnRemoved)) {
169         canvas.sendRemovedCards(cardStackRemovedCards.getRemovedCards(↵
txtDiceNotation.getText()));
170         canvas.ACTION_DRAW = Canvas.DRAW_FOR_REMOVED;
171     } else if (btnSrc.equals(btnFutureCards)) {
172         canvas.sendFutureCards(collectionCardStacks.getFutureCards(txtDiceNotation.↵
getText()));
173         canvas.ACTION_DRAW = Canvas.DRAW_FOR_FUTURE;
174     } else if (btnSrc.equals(btnShuffle)) {
175         if (collectionCardStacks.shuffleStack(txtDiceNotation.getText())) {
176             txtNotification.append("\n\nShuffling stack " + txtDiceNotation.getText()↵
+ " complete.");
177             canvas.repaint();
178         } else {
179             txtNotification.append("\n\nNo such stack to shuffle.");
180         }
181     }
182 }
183
184 @Override
185 public void send(String notification
186 ) {
187     txtNotification.append(notification);
188 }
189
190 public class MyButton extends JButton {
191
192     MyButton(String text) {
193         setText(text);
194         setFont(new Font("Century Gothic", Font.BOLD, 14));
195         setBackground(Color.decode("#B38B6D"));
196         setFocusPainted(false);
197         setBorder(BorderFactory.createCompoundBorder(BorderFactory.createBevelBorder(↵
(BevelBorder.RAISED), BorderFactory.createEmptyBorder(5, 5, 5, 5)));
198     }
199 }
200
201 public static void main(String[] args) {
202     UserInterface UI = new UserInterface();
203     UI.setVisible(true);
204 }
205 }
206

```

Inner Class: MyButton

```
190 public class MyButton extends JButton {
191
192     MyButton(String text) {
193         setText(text);
194         setFont(new Font("Century Gothic", Font.BOLD, 14));
195         setBackground(Color.decode("#B38B6D"));
196         setFocusPainted(false);
197         setBorder(BorderFactory.createCompoundBorder(↵
BorderFactory.createBevelBorder(BevelBorder.RAISED), BorderFactory.↵
createEmptyBorder(5, 5, 5, 5)));
198     }
199 }
200
```

Nested inside Class: UserInterface.

Interface: NotificationListener

```
6 package cardstacks;
7
8 /**
9  *
10  * @author Biju Ale
11  */
12 public interface NotificationListener {
13     public void send(String notification);
14 }
15
```

Interface: DrawActionListener

```
6   package cardstacks;
7
8   import java.util.ArrayList;
9
10  /**
11   *
12   * @author Biju Ale
13   */
14  public interface DrawActionListener {
15
16      public void sendAllDealtCards(ArrayList allDealtCards);
17
18      public void sendSingleDealtCard(Card singleDealtCard);
19
20      public void sendRemovedCards(ArrayList allRemovedCards);
21
22      public void sendFutureCards(CardStack futureCards);
23
24  }
25
```

Class: Canvas

[PLEASE TURN OVER]

```

6   package cardstacks;
7
8   import java.awt.Font;
9   import java.awt.Graphics;
10  import java.awt.Graphics2D;
11  import java.awt.RenderingHints;
12  import java.awt.geom.AffineTransform;
13  import java.awt.geom.Rectangle2D;
14  import java.util.ArrayList;
15  import java.util.Collection;
16  import javax.swing.JPanel;
17
18  /**
19   *
20   * @author Biju Ale
21   */
22  public class Canvas extends JPanel implements DrawActionListener {
23
24      public int ACTION_DRAW;
25      public final static int DRAW_FOR_GET_CARD = 1;
26      public final static int DRAW_FOR_DEALT_CARD = 2;
27      public final static int DRAW_FOR_REMOVED = 3;
28      public final static int DRAW_FOR_FUTURE = 4;
29
30      private ArrayList<Card> allDealtCards;
31      private ArrayList<Card> allRemovedCards;
32      private Card singleDealtCard;
33      private CardStack futureCards;
34
35      private int x = 5, y = 25;
36      NotificationListener notificationListener;
37
38      @Override
39      protected void paintComponent(Graphics g) {
40          super.paintComponent(g);
41          Graphics2D g2d = (Graphics2D) g;
42          g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_
ANTIALIAS_ON);
43          switch (ACTION_DRAW) {
44              case DRAW_FOR_GET_CARD:
45                  drawForGetCard(g2d);

```



```

46         break;
47     case DRAW_FOR_REMOVED:
48         drawForPeekRemovedCard(g2d);
49         break;
50     case DRAW_FOR_DEALT_CARD:
51         drawForPeekDealtCard(g2d);
52         break;
53     case DRAW_FOR_FUTURE:
54         drawForPeekFutureCard(g2d);
55         break;
56     }
57 }
58
59 private void drawForGetCard(Graphics g2d) {
60     if (singleDealtCard.getDiceNotation() != null) {
61         notificationListener.send("\n\nGetting card...");
62         g2d.setFont(new Font("Times New Roman", Font.PLAIN, 25));
63         g2d.drawString("Recently dealt card for stack: " + singleDealtCard.↵
getDiceNotation(), x, y);
64
65         Font font = new Font("Times New Roman", Font.PLAIN, 200);
66         g2d.setFont(font);
67         String num = "" + singleDealtCard.getNumber();
68         g2d.drawString(num, 250, 292);
69
70         Font font1 = new Font("Times New Roman", Font.PLAIN, 50);
71         g2d.setFont(font1);
72         String freq = "" + (double) Math.round((singleDealtCard.getFrequency() / ↵
100d) * 10d) / 10d + "%";
73         g2d.drawString(freq, 250, 394);
74     } else {
75         notificationListener.send("\n\nNo card exists to deal! Enter valid notation"↵
);
76     }
77 }
78
79 private void drawForPeekDealtCard(Graphics g2d) {
80     if (allDealtCards != null && allDealtCards.size() > 0) {
81         notificationListener.send("\n\nGetting dealt cards with Graph...");
82         int x = 5;

```

```

83         g2d.setFont(new Font("Times New Roman", Font.PLAIN, 20));
84         g2d.drawString("All Dealt Cards from stack: " + allDealtCards.get(0).↵
getDiceNotation(), x, y);
85
86         for (Card eachDealtCard : allDealtCards) {
87             g2d.drawString("" + eachDealtCard.getNumber() + " [" + (double) Math.↵
round((eachDealtCard.getFrequency() / 100d) * 10d) / 10d + "%], ", x, 50);
88             x += 90;
89         }
90         drawGraph(g2d, "Dealt Cards", allDealtCards);
91     } else {
92         notificationListener.send("\n\nNo dealt card exists to peek!");
93     }
94 }
95
96 private void drawForPeekRemovedCard(Graphics g2d) {
97     if (allRemovedCards != null && allRemovedCards.size() > 0) {
98         notificationListener.send("\n\nGetting removed cards...");
99         int x = 5;
100        int y = 250;
101        g2d.setFont(new Font("Times New Roman", Font.PLAIN, 25));
102        g2d.drawString("All Removed Cards from stack: " + allRemovedCards.get(0).↵
getDiceNotation(), x, y);
103
104        for (Card eachRemovedCard : allRemovedCards) {
105            g2d.drawString("" + eachRemovedCard.getNumber() + " [" + (double) Math.↵
round((eachRemovedCard.getFrequency() / 100d) * 10d) / 10d + "%], ", x, y + 40);
106            x += 110;
107        }
108    } else {
109        notificationListener.send("\n\nNo removed card exists to peek!");
110    }
111 }
112
113 private void drawForPeekFutureCard(Graphics g2d) {
114     if (futureCards != null && futureCards.size() > 0) {
115         notificationListener.send("\n\nGetting future cards with Graph...");
116         int x = 5;
117        g2d.setFont(new Font("Times New Roman", Font.PLAIN, 20));
118        g2d.drawString("Future Cards from stack: " + futureCards.getFirst().↵
getDiceNotation(), x, y);

```

```

119
120         for (Card eachFutureCard : futureCards) {
121             g2d.drawString("" + eachFutureCard.getNumber() + " [" + (double) Math.↵
round((eachFutureCard.getFrequency() / 100d) * 10d) / 10d + "%]", "", x, 50);
122             x += 90;
123         }
124         drawGraph(g2d, "Future Cards", futureCards);
125     } else {
126         notificationListener.send("\n\nNo future cards exists left to peek!");
127     }
128
129 }
130
131 private void drawGraph(Graphics g, String cardType, Collection<Card> stack) {
132     Graphics2D g2d = (Graphics2D) g;
133     g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_↵
ANTIALIAS_ON);
134     int x = 40, y = 100;
135
136     //Y-Coordinate lines
137     int s = y;
138     for (int i = y; i > -10; i = i - 10) {
139         g2d.drawString("" + (i), x - 10, s + 8);
140         g2d.drawLine(x + 20, s, getWidth(), s);
141         s += 40;
142     }
143
144     //X-Coordinate label
145     g2d.setFont(new Font("Times New Roman", Font.PLAIN, 20));
146     // g2d.setColor(Color.decode("#371C00"));
147     g2d.drawString(cardType + " (Card number)", 350, 550);
148
149     //Y-Coordinate label
150     AffineTransform t = new AffineTransform();
151     t.rotate(Math.toRadians(-90));
152     g2d.setFont(new Font("Times New Roman", Font.PLAIN, 20).deriveFont(t));
153     // g2d.setColor(Color.decode("#371C00"));
154     g2d.drawString("Chance of occurence (%)", x - 17, 380);
155
156     //Bars
157     double chartHeight = 400;
158     double barWidth = 15;

```

```

159         for (Card eachCard : stack) {
160             double barHeight = (double) eachCard.getFrequency() / (double) Dice.ROLL_ ←
TIMES * 100 * 4d;
161             double newY = chartHeight - barHeight;
162
163             Rectangle2D bar = new Rectangle2D.Double(x + 50, newY + y, barWidth, ←
barHeight);
164             g2d.fill(bar);
165             AffineTransform t1 = new AffineTransform();
166             t1.rotate(Math.toRadians(-90));
167             g2d.setFont(new Font("Times New Roman", Font.ITALIC, 12).deriveFont(t1));
168             g2d.drawString("" + eachCard.getNumber(), (float) x + 50, (float) ←
chartHeight + 120);
169             x += 30;
170         }
171     }
172
173     @Override
174     public void sendAllDealtCards(ArrayList allDealtCards) {
175         this.allDealtCards = allDealtCards;
176         this.repaint();
177     }
178
179     @Override
180     public void sendSingleDealtCard(Card singleDealtCard) {
181         this.singleDealtCard = singleDealtCard;
182         this.repaint();
183     }
184
185     @Override
186     public void sendRemovedCards(ArrayList allRemovedCards) {
187         this.allRemovedCards = allRemovedCards;
188         this.repaint();
189     }
190
191     @Override
192     public void sendFutureCards(CardStack futureCards) {
193         this.futureCards = futureCards;
194         this.repaint();
195     }
196
197     public void addNotificationListener(NotificationListener notificationListener) {
198         this.notificationListener = notificationListener;
199     }
200 }
201

```

Task-2 (Testing Data)

Test Suite No.1

Testingclass: cardstacks.NotationsReader

Testingtype: Black Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	void parseDiceNotation(String diceNotation)	4d4-2	Valid (2 delimiters – 'd' & '-')	Sets values of instance variables as following: <ul style="list-style-type: none">• diceNotation = 4d4-2• numDices = 4• numFaces = 4• toRemove = 2
2	void parseDiceNotation(String diceNotation)	4d4	Valid (1 delimiter – 'd')	Sets values instance variables as following: <ul style="list-style-type: none">• diceNotation = 4d4• numDices = 4• numFaces = 4• toRemove = 0
3	void parseDiceNotation(String diceNotation)	xyz	Invalid	Throws NumberFormatException with message - "\n\nINVALID DICE NOTATION!"
4	void parseDiceNotation(String diceNotation)	""	Null	Throws NumberFormatException with message - "\n\nINVALID DICE NOTATION!"

Method of Equivalence partitioning:

- Two types of valid (validated by Regex Pattern matcher) input is present – 1 sample was selected from each equivalence partition.
- Anything besides valid input's Regex Pattern is another partition. 1 sample was selected.
- If no input is given, this is taken as another partition. Null is selected.
- No boundary value analysis required as per the nature of expected parameter.

Test Execution

Source Code

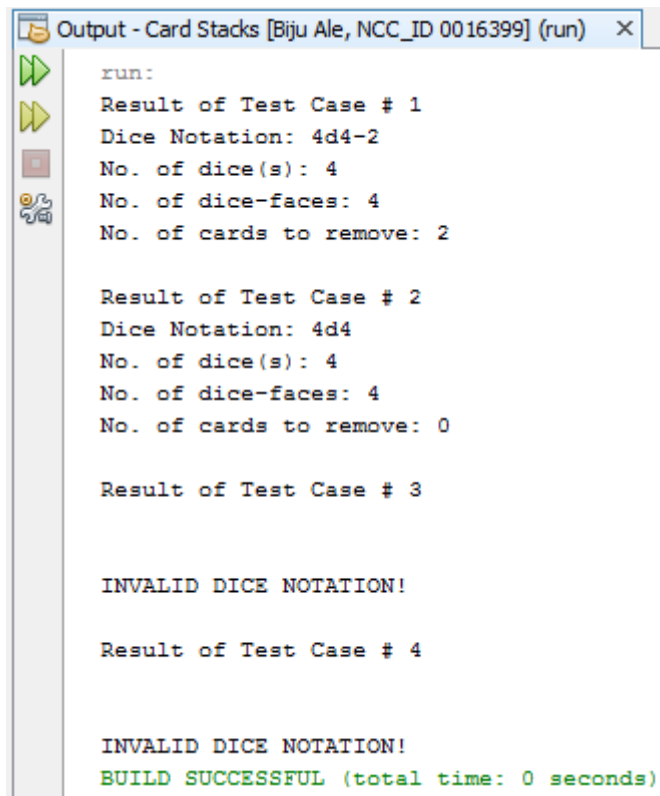
[PLEASE TURN OVER]

```

6   package test;
7
8   import cardstacks.NotationTokenReader;
9
10  /**
11   *
12   * @author Biju Ale
13   */
14  public class BlackBxTestANotationReader {
15
16      public static void main(String[] args) {
17          NotationTokenReader nreader = new NotationTokenReader();
18          System.out.println("Result of Test Case # 1");
19          try {
20              nreader.parseDiceNotation("4d4-2");
21              System.out.println("Dice Notation: " + nreader.getDiceNotation());
22              System.out.println("No. of dice(s): " + nreader.getNumDices());
23              System.out.println("No. of dice-faces: " + nreader.getNumFaces());
24              System.out.println("No. of cards to remove: " + nreader.getToRemove());
25          } catch (Exception ex) {
26              System.out.println(ex.getMessage());
27          }
28
29          System.out.println("\nResult of Test Case # 2");
30          try {
31              nreader.parseDiceNotation("4d4");
32              System.out.println("Dice Notation: " + nreader.getDiceNotation());
33              System.out.println("No. of dice(s): " + nreader.getNumDices());
34              System.out.println("No. of dice-faces: " + nreader.getNumFaces());
35              System.out.println("No. of cards to remove: " + nreader.getToRemove());
36          } catch (Exception ex) {
37              System.out.println(ex.getMessage());
38          }
39
40          System.out.println("\nResult of Test Case # 3");
41          try {
42              nreader.parseDiceNotation("xyz");
43          } catch (Exception ex) {
44              System.out.println(ex.getMessage());
45          }
46
47          System.out.println("\nResult of Test Case # 4");
48          try {
49              nreader.parseDiceNotation("");
50          } catch (Exception ex) {
51              System.out.println(ex.getMessage());
52          }
53      }
54  }
55

```

Output



```
run:
Result of Test Case # 1
Dice Notation: 4d4-2
No. of dice(s): 4
No. of dice-faces: 4
No. of cards to remove: 2

Result of Test Case # 2
Dice Notation: 4d4
No. of dice(s): 4
No. of dice-faces: 4
No. of cards to remove: 0

Result of Test Case # 3

INVALID DICE NOTATION!

Result of Test Case # 4

INVALID DICE NOTATION!
BUILD SUCCESSFUL (total time: 0 seconds)
```


Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	void parseDiceNotation(String diceNotation)	4d4- 2	Valid (2 delimiters – 'd' & '-')	Sets values of instance variables as following: <ul style="list-style-type: none">• diceNotation = 4d4-2• numDices = 4• numFaces = 4• toRemove = 2	Yes
2	void parseDiceNotation(String diceNotation)	4d4	Valid (1 delimiter – 'd')	Sets values instance variables as following: <ul style="list-style-type: none">• diceNotation = 4d4• numDices = 4• numFaces = 4• toRemove = 0	Yes
3	void parseDiceNotation(String diceNotation)	xyz	Invalid	Throws NumberFormatException with message - "\n\nINVALID DICE NOTATION!"	Yes
4	void parseDiceNotation(String diceNotation)	""	Null	Throws NumberFormatException with message - "\n\nINVALID DICE NOTATION!"	Yes

Test Summary

From the above test results, all tests were executed as expected.

Test Suite No.1 also implicitly covered white box & black box tests for getter methods, which returned the respective values of instance variables. Hence, it too executed as expected without any errors.

Test Suite No.2

Testingclass: cardstacks.NotationsReader

Testingtype: White Box / Unit Testing

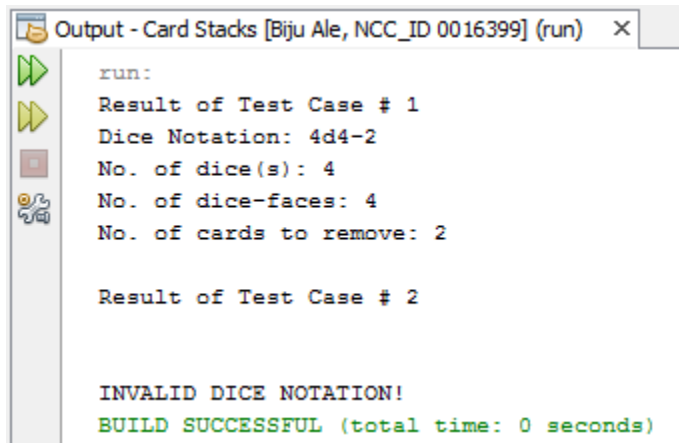
#	Method	Test Data	Input type	Expected Outcome
1	void parseDiceNotation(String diceNotation)	4d4- 2	Valid (2 delimiters – 'd' & '-')	Sets values of instance variables as following: <ul style="list-style-type: none">• diceNotation = 4d4-2• numDices = 4• numFaces = 4• toRemove = 2
2	void parseDiceNotation(String diceNotation)	xyz	Invalid	Throws NumberFormatException with message - "INVALID DICE NOTATION!"

Test Execution

Source Code

```
6   package test;
7
8   import cardstacks.NotationTokenReader;
9
10  /**
11   *
12   * @author Biju Ale
13   */
14  public class WhiteBxTestANotationReader {
15
16      public static void main(String[] args) {
17          NotationTokenReader nreader = new NotationTokenReader();
18
19          System.out.println("Result of Test Case # 1");
20          try {
21              nreader.parseDiceNotation("4d4-2");
22              System.out.println("Dice Notation: " + nreader.getDiceNotation());
23              System.out.println("No. of dice(s): " + nreader.getNumDices());
24              System.out.println("No. of dice-faces: " + nreader.getNumFaces());
25              System.out.println("No. of cards to remove: " + nreader.getToRemove());
26          } catch (Exception ex) {
27              System.out.println(ex.getMessage());
28          }
29
30          System.out.println("\nResult of Test Case # 2");
31          try {
32              nreader.parseDiceNotation("xyz");
33          } catch (Exception ex) {
34              System.out.println(ex.getMessage());
35          }
36      }
37  }
```

Output



```
run:
Result of Test Case # 1
Dice Notation: 4d4-2
No. of dice(s): 4
No. of dice-faces: 4
No. of cards to remove: 2

Result of Test Case # 2

INVALID DICE NOTATION!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

Test Suite No.3

Testingclass: cardstacks.Dice

Testingtype: Black Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	Dice(NotificationReader nreader)	nreader	Valid object	Constructor should set the instance variable value i.e. value of dice name. It should also invoke complementary private methods: setMinMax method which invokes populateCombinations method which invokes roll method.
2	Dice (NotificationReader nreader)	nreader	*Invalid object (due to invalid data member)	Constructor should not set the dice name. Exception should be thrown with message – "No. of cards to remove cannot exceed total no. of cards. Enter valid notation.\n"

Method of test data selection:

- *Here, invalid data member of 'nreader' means the parsing was correct (Test Suite No.1) but the number of card to remove exceeded the maximum combination number (total no. of cards). Correct parsing is checked in Test Suite No.1 whereas the valid no. of card to remove is checked in Test Suite No.3's constructor.
- It never receives null as input, because before the constructor is called, null is already validated by NotationReader that was checked in test suite no.1 and 2.

Test Execution

Source Code

[PLEASE TURN OVER]

```

6   package test;
7
8   import cardstacks.Dice;
9   import cardstacks.NotationTokenReader;
10
11  /**
12   *
13   * @author Biju Ale
14   */
15  public class BlackBxTestADice {
16
17      public static void main(String[] args) {
18          System.out.println("Result of Test Case #1");
19          NotationTokenReader nreader = new NotationTokenReader();
20          try {
21              nreader.parseDiceNotation("4d4");
22              Dice dice = new Dice(nreader);
23              //Testing Constructor
24              System.out.println("Result of Constructor:");
25              System.out.println("Checking no. of card to remove is < max combination no.↵
26              .:\tYes");
27              System.out.println("Dice Object created.");
28              System.out.println("Dice name set to: " + dice.getDiceName());
29
30              //Testing setMinMax invoked by constructor
31              System.out.println("\nResult of setMinMax invoked by constructor:");
32              System.out.println("Minimum Card number: " + dice.getMinCombination());
33              System.out.println("Maximum Card number: " + dice.getMaxCombination());
34
35              //Testing populateCombinations invoked by setMinMax
36              System.out.println("\nResult of populateCombinations invoked by setMinMax:↵
37              ");
38              System.out.println("All possible combinations of a 4d4 dice");
39              for (Integer eachCombination : dice.getCombinations()) {
40                  System.out.println(eachCombination);
41              }
42
43              //Testing roll invoked by populateCombinations
44              System.out.println("\nResult of roll invoked by populateCombinations:");
45              int sumFrequencies = 0;
46              for (int i = 0; i < dice.getFrequencies().length; i++) {

```

```

45         System.out.println("Card: " + dice.getCombinations()[i] + " Frequency↵
: " + dice.getFrequencies()[i]);
46         sumFrequencies += dice.getFrequencies()[i];
47     }
48     System.out.println("Frequency Sum: " + sumFrequencies);
49 } catch (Exception ex) {
50     System.out.println(ex.getMessage());
51 }
52 System.out.println("\nResult of Test Case #2");
53 try {
54     nreader.parseDiceNotation("4d4-344");
55     Dice dice = new Dice(nreader);
56 } catch (Exception ex) {
57     System.out.println("Checking no. of card to remove is < max combination no↵
.:\tNo");
58     System.out.println("Dice name not set: ");
59     System.out.println(ex.getMessage());
60 }
61 }
62
63 }
64

```

Output

```
Output - Card Stacks [Biju Ale, NCC_ID 0016399] (run) X
run:
Result of Test Case #1
Result of Constructor:
Checking no. of card to remove is < max combination no.:      Yes
Dice Object created.
Dice name set to: 4d4

Result of setMinMax invoked by constructor:
Minimum Card number: 4
Maximum Card number: 16

Result of populateCombinations invoked by setMinMax:
All possible combinations of a 4d4 dice
4
5
6
7
8
9
10
11
12
13
14
15
16

Result of roll invoked by populateCombinations:
Card: 4 Frequency: 773
Card: 5 Frequency: 790
Card: 6 Frequency: 773
Card: 7 Frequency: 740
Card: 8 Frequency: 762
Card: 9 Frequency: 785
Card: 10 Frequency: 841
Card: 11 Frequency: 766
Card: 12 Frequency: 779
Card: 13 Frequency: 686
Card: 14 Frequency: 744
Card: 15 Frequency: 763
Card: 16 Frequency: 798
Frequency Sum: 10000
```



```

Result of Test Case #2
Checking no. of card to remove is < max combination no.:      No
Dice name not set:
No. of cards to remove cannot exceed total no. of cards. Enter valid notation.

BUILD SUCCESSFUL (total time: 2 seconds)

```

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	Dice(NotificationReader nreader)	nreader	Valid object	<p>Constructor should set the instance variable value i.e. value of dice name.</p> <p>.</p> <p>It should also invoke complementary private methods: setMinMax method which invokes populateCombinations method which invokes roll method.</p>	Yes
2	Dice (NotificationReader nreader)	nreader	*Valid object (with invalid data member)	<p>Constructor should not set the dice name.</p> <p>Exception should be thrown with message – "No. of cards to remove cannot exceed total no. of cards. Enter valid notation.\n"</p>	Yes

Test Summary

Since, testing constructor invoked 3 other complementary private methods. It is safe to say that the 3 additional methods were implicitly black-box tested.

From the above test results, all tests were executed as expected.

Test Suite No.4

Testingclass: cardstacks.Dice

Testingtype: White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	Dice(NotitficationReader nreader)	nreader	Valid object	Constructor should set the instance variable value i.e. value of dice name. It should also invoke complementary private methods: setMinMax method which invokes populateCombinations method which invokes roll method.
2	Dice (NotificationReader nreader)	nreader	*Invalid object (due to invalid data member)	Constructor should not set the dice name. Exception should be thrown with message – "No. of cards to remove cannot exceed total no. of cards. Enter valid notation.\n"

Method of test data selection:

- *Here, invalid data member of 'nreader' the parsing was correct correct (Test Suite No.1) but the number of card to remove exceeded the maximum combination number (total no. of cards). Correct parsing is checked in Test Suite No.1 whereas the valid no. of card to remove is checked in Test Suite No.3's constructor.
- It never receives null as input, because before the constructor is called, null is already validated by NotationReader that was checked in test suite no.1 and 2.

Test Execution

Source Code

[PLEASE TURN OVER]

```

6   package test;
7
8   import cardstacks.Dice;
9   import cardstacks.NotationTokenReader;
10
11  /**
12   *
13   * @author Biju Ale
14   */
15  public class WhiteBxTestADice {
16
17      public static void main(String[] args) {
18          System.out.println("Result of Test Case #1");
19          NotationTokenReader nreader = new NotationTokenReader();
20          try {
21              nreader.parseDiceNotation("4d4");
22              Dice dice = new Dice(nreader);
23              //Testing Constructor
24              System.out.println("Result of Constructor:");
25              System.out.println("Checking no. of card to remove is < max combination no.↵
26  .:\tYes");
27              System.out.println("Dice Object created.");
28              System.out.println("Dice name set to: " + dice.getDiceName());
29
30              //Testing setMinMax invoked by constructor
31              System.out.println("\nResult of setMinMax invoked by constructor:");
32              System.out.println("Minimum Card number: " + dice.getMinCombination());
33              System.out.println("Maximum Card number: " + dice.getMaxCombination());
34
35              //Testing populateCombinations invoked by setMinMax
36              System.out.println("\nResult of populateCombinations invoked by setMinMax:↵
37  ");
38              System.out.println("All possible combinations of a 4d4 dice");
39              for (Integer eachCombination : dice.getCombinations()) {
40                  System.out.println(eachCombination);
41              }
42          }
43      }
44  }

```

```

41 //Testing roll invoked by populateCombinations
42 System.out.println("\nResult of roll invoked by populateCombinations:");
43 int sumFrequencies = 0;
44 for (int i = 0; i < dice.getFrequencies().length; i++) {
45     System.out.println("Card: " + dice.getCombinations()[i] + " Frequency↵
: " + dice.getFrequencies()[i]);
46     sumFrequencies += dice.getFrequencies()[i];
47 }
48 System.out.println("Frequency Sum: " + sumFrequencies);
49 } catch (Exception ex) {
50     System.out.println(ex.getMessage());
51 }
52 System.out.println("\nResult of Test Case #2");
53 try {
54     nreader.parseDiceNotation("4d4-344");
55     Dice dice = new Dice(nreader);
56 } catch (Exception ex) {
57     System.out.println("Checking no. of card to remove is < max combination no↵
.:\tNo");
58     System.out.println("Dice name not set: ");
59     System.out.println(ex.getMessage());
60 }
61 }
62 }

```

Output:

[PLEASE TURN OVER]

```
Output - Card Stacks [Biju Ale, NCC_ID 0016399] (run) X
run:
Result of Test Case #1
Result of Constructor:
Checking no. of card to remove is < max combination no.:      Yes
Dice Object created.
Dice name set to: 4d4

Result of setMinMax invoked by constructor:
Minimum Card number: 4
Maximum Card number: 16

Result of populateCombinations invoked by setMinMax:
All possible combinations of a 4d4 dice
4
5
6
7
8
9
10
11
12
13
14
15
16

Result of roll invoked by populateCombinations:
Card: 4 Frequency: 773
Card: 5 Frequency: 790
Card: 6 Frequency: 773
Card: 7 Frequency: 740
Card: 8 Frequency: 762
Card: 9 Frequency: 785
Card: 10 Frequency: 841
Card: 11 Frequency: 766
Card: 12 Frequency: 779
Card: 13 Frequency: 686
Card: 14 Frequency: 744
Card: 15 Frequency: 763
Card: 16 Frequency: 798
Frequency Sum: 10000

Result of Test Case #2
Checking no. of card to remove is < max combination no.:      No
Dice name not set:
No. of cards to remove cannot exceed total no. of cards. Enter valid notation.

BUILD SUCCESSFUL (total time: 2 seconds)
```

Test Summary

Since, testing constructor invoked 3 other complementary private methods. It is safe to say that the 3 additional methods were implicitly white-box tested.

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

Test Suite No.5

Testingclass: cardstacks.Dice

Testingtype: Black Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	Card(int number, int frequency, String diceNotation)	(4,453,"4d4")	Valid	Constructor should set the values of instance variables as following: <ul style="list-style-type: none">• number = 4• frequency = 453• diceNotation = 4d4

Method of test data selection:

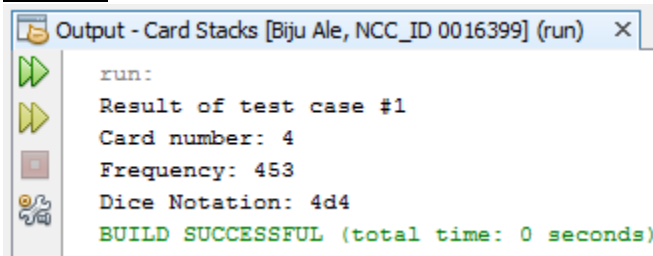
This constructor always receives valid input only, because it is only invoked after NotationReader class that was checked in Test Suite No.1 & 2, validates the input from GUI.

Test Execution

Source Code

```
6  package test;
7
8  import cardstacks.Card;
9
10 /**
11  *
12  * @author Biju Ale
13  */
14 public class BlackBxTestACard {
15
16     public static void main(String[] args) {
17         System.out.println("Result of test case #1");
18         Card card = new Card(4, 453, "4d4");
19         System.out.println("Card number: " + card.getNumber());
20         System.out.println("Frequency: " + card.getFrequency());
21         System.out.println("Dice Notation: " + card.getDiceNotation());
22     }
23 }
```

Output



```
run:
Result of test case #1
Card number: 4
Frequency: 453
Dice Notation: 4d4
BUILD SUCCESSFUL (total time: 0 seconds)
```


Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	Card(int number, int frequency, String diceNotation)	(4,453,"4d4")	Valid	Constructor should set the values of instance variables as following: <ul style="list-style-type: none">• number = 4• frequency = 453• diceNotation = 4d4	Yes

Test Summary

From the above test results, all tests were executed as expected.

Test Suite No.6

Testingclass: cardstacks.Dice

Testingtype: White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	Card(int number, int frequency, String diceNotation)	(4,453,"4d4")	Valid	Constructor should set the values of instance variables as following: <ul style="list-style-type: none">• number = 4• frequency = 453• diceNotation = 4d4

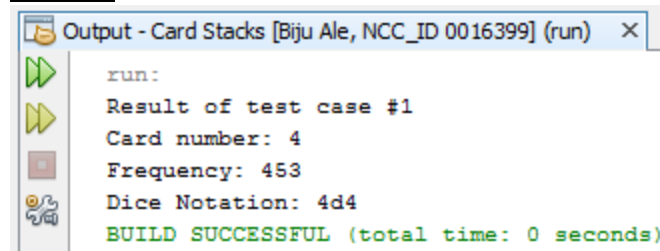
Method of test data selection:

This constructor always receives valid input only, because it is only invoked after NotationReader calss that was checked in Test Suite No.1 & 2, validates the input from GUI.

Test Execution

```
6 package test;
7
8 import cardstacks.Card;
9
10 /**
11  *
12  * @author Biju Ale
13  */
14 public class WhiteBxTestACard {
15
16     public static void main(String[] args) {
17         System.out.println("Result of test case #1");
18         Card card = new Card(4, 453, "4d4");
19         System.out.println("Card number: " + card.getNumber());
20         System.out.println("Frequency: " + card.getFrequency());
21         System.out.println("Dice Notation: " + card.getDiceNotation());
22     }
23 }
24
```

Output



Output - Card Stacks [Biju Ale, NCC_ID 0016399] (run) X

```
run:
Result of test case #1
Card number: 4
Frequency: 453
Dice Notation: 4d4
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

Test Suite No.7

Testingclass: cardstacks.CardStack

Testingtype: Black Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	CardStack(Dice dice, NotationReader nreader, CardStackRemovedCards csrc)	*(dice, nreader, csrc)	Valid objects	Constructor should set the instance variable value i.e. value of dice notation. It should also invoke populateCardStack method which will add all shuffled cards to CardStack.
2	CardStack(Dice dice, NotationReader nreader, CardStackRemovedCards csrc)	(dice,nreader, csrc)	**Valid objects	Constructor should set the instance variable value i.e. value of dice notation. It should also invoke populateCardStack method which will add all shuffled cards to CardStack. It should then remove the correct no. of cards.

Method of test data selection:

- *Object of CardStackRemovedCards csrc is instantiated in GUI using default constructor. It does not contain any data member/instance variables. No. of card to remove here, is 0.
- **'nreader' contains data member, where card to remove is greater than 0.
- It never receives null as input, because before the constructor is called, null is already validated by NotationReader that was checked in test suite no.1 and 2.

Test Execution

Source Code

[PLEASE TURN OVER]

```

6 package test;
7
8 import cardstacks.CardStack;
9 import cardstacks.CardStackRemovedCards;
10 import cardstacks.Dice;
11 import cardstacks.NotationTokenReader;
12
13 /**
14  *
15  * @author Biju Ale
16  */
17 public class BlackBxTestCardStack {
18
19     public static void main(String[] args) {
20         try {
21             System.out.println("Result of test case #1");
22             NotationTokenReader nreader = new NotationTokenReader();
23             nreader.parseDiceNotation("2d2");
24             Dice dice = new Dice(nreader);
25             CardStackRemovedCards csrc = new CardStackRemovedCards();
26
27             CardStack cardStack = new CardStack(dice, nreader, csrc);
28             System.out.println("diceNotation: " + cardStack.getDiceNotation());
29             System.out.println("Cards to remove: " + nreader.getToRemove());
30
31         } catch (Exception ex) {
32             System.out.println(ex.getMessage());
33         }
34         try {
35             System.out.println("\nResult of test case #2");
36             NotationTokenReader nreader = new NotationTokenReader();
37             nreader.parseDiceNotation("2d2-1");
38             Dice dice = new Dice(nreader);
39             CardStackRemovedCards csrc = new CardStackRemovedCards();
40
41             CardStack cardStack = new CardStack(dice, nreader, csrc);
42             System.out.println("diceNotation: " + cardStack.getDiceNotation());
43             System.out.println("Cards to remove: " + nreader.getToRemove());
44

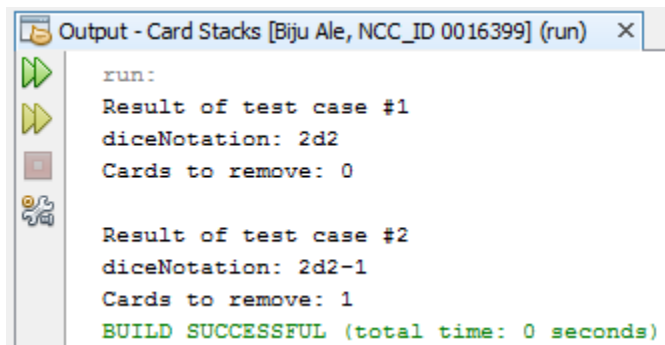
```

```

45         } catch (Exception ex) {
46             System.out.println(ex.getMessage());
47         }
48     }
49 }
50

```

Output



The screenshot shows an IDE's Output window titled "Output - Card Stacks [Biju Ale, NCC_ID 0016399] (run)". The window contains the following text:

```

run:
Result of test case #1
diceNotation: 2d2
Cards to remove: 0

Result of test case #2
diceNotation: 2d2-1
Cards to remove: 1
BUILD SUCCESSFUL (total time: 0 seconds)

```

After constructor invoked populateCardStack & removeCard methods, it added shuffled cards to CardStack.

Since, there is no getter for CardStack's elements that test package can access, debug mode was used to test if populateCardStack was invoked by constructor with correct results. Following are the results:

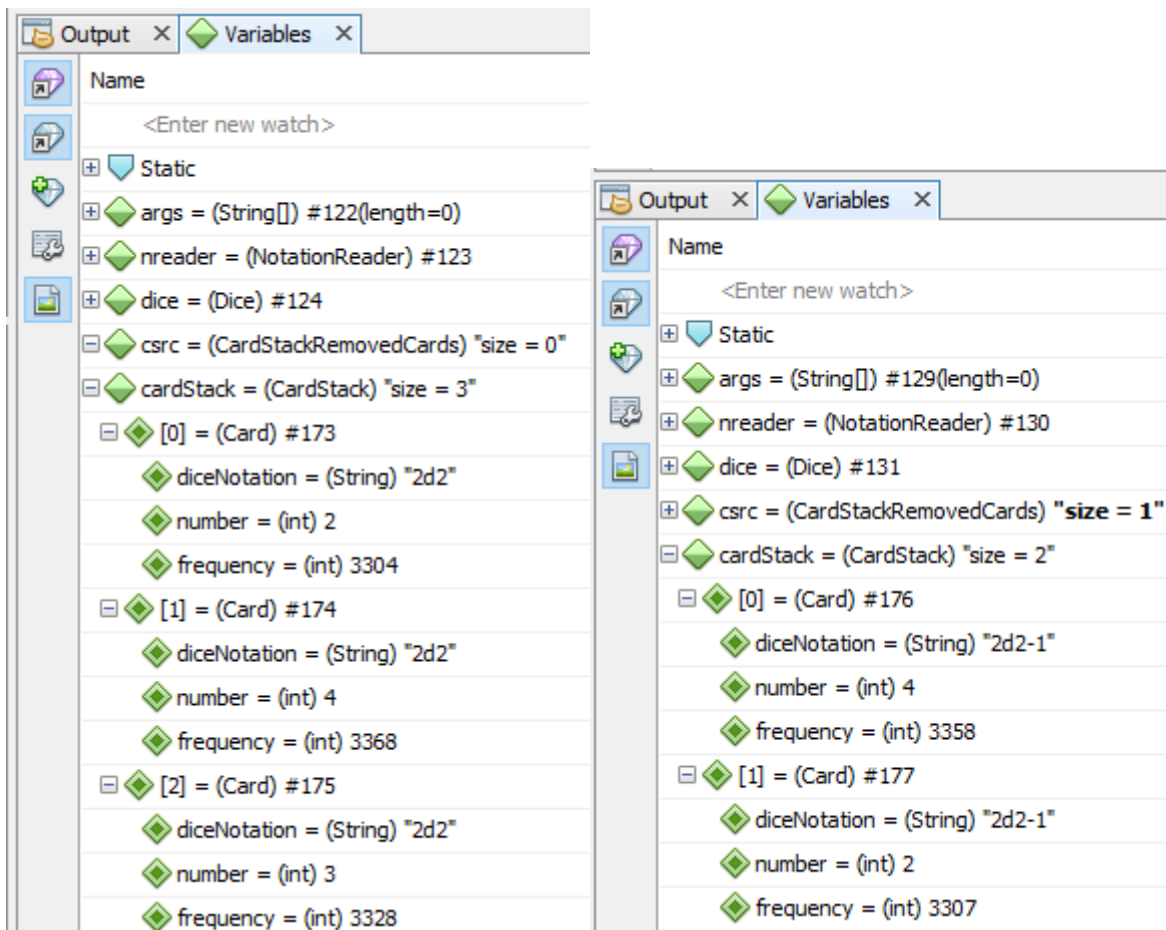


Figure: Checking if populateCardStack invoked by constructor removed correct no. of cards. For test case #1 (left) & test case#2.

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	CardStack(Dice dice, NotationReader nreader, CardStackRemovedCards csrc)	*(dice, nreader, csrc)	Valid objects	Constructor should set the instance variable value i.e. value of dice notation. It should also invoke populateCardStack method which will add all shuffled cards to CardStack.	Yes
2	CardStack(Dice dice, NotationReader nreader, CardStackRemovedCards csrc)	(dice,nreader, csrc)	**Valid objects	Constructor should set the instance variable value i.e. value of dice notation. It should also invoke populateCardStack method which will add all shuffled cards to CardStack. It should then remove the correct no. of cards.	Yes

Test Summary

Since, testing constructor invoked 2 other complementary private methods. It is safe to say that the 2 additional methods were implicitly black-box tested.

From the above test results, all tests were executed as expected.

Test Suite No.8

Testingclass: cardstacks.CardStack

Testingtype: White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	CardStack(Dice dice, NotationReader nreader, CardStackRemovedCards csrc)	*(dice, nreader, csrc)	Valid objects	Constructor should set the instance variable value i.e. value of dice notation. It should also invoke populateCardStack & removeCard private methods which will add all shuffled cards to CardStack.
2	CardStack(Dice dice, NotationReader nreader, CardStackRemovedCards csrc)	(dice,nreader, csrc)	**Valid objects	Constructor should set the instance variable value i.e. value of dice notation. It should also invoke populateCardStack & removeCard private methods which will add all shuffled cards to CardStack. It should then remove the correct no. of cards.

Method of test data selection:

- *Object of CardStackRemovedCards csrc is instantiated in GUI using default constructor. It does not contain any data member/instance variables. No. of card to remove here, is 0.
- **'nreader' contains data member, where card to remove is greater than 0.
- It never receives null as input, because before the constructor is called, null is already validated by NotationReader that was checked in test suite no.1 and 2.

Test Execution

Source Code

[PLEASE TURN OVER]

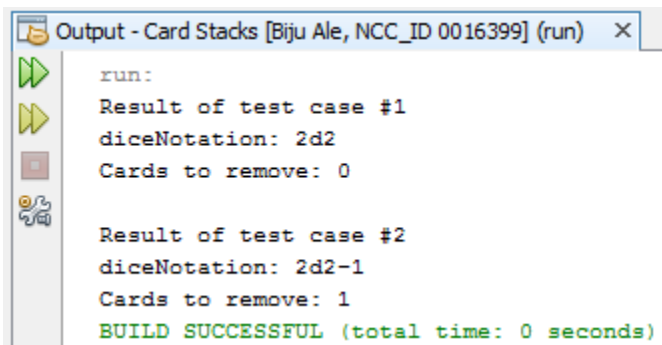
```

6 package test;
7
8 import cardstacks.CardStack;
9 import cardstacks.CardStackRemovedCards;
10 import cardstacks.Dice;
11 import cardstacks.NotationTokenReader;
12
13 /**
14  *
15  * @author Biju Ale
16  */
17 public class WhiteBxTestCardStack {
18
19     public static void main(String[] args) {
20         try {
21             System.out.println("Result of test case #1");
22             NotationTokenReader nreader = new NotationTokenReader();
23             nreader.parseDiceNotation("2d2");
24             Dice dice = new Dice(nreader);
25             CardStackRemovedCards csrc = new CardStackRemovedCards();
26
27             CardStack cardStack = new CardStack(dice, nreader, csrc);
28             System.out.println("diceNotation: " + cardStack.getDiceNotation());
29             System.out.println("Cards to remove: " + nreader.getToRemove());
30
31         } catch (Exception ex) {
32             System.out.println(ex.getMessage());
33         }
34         try {
35             System.out.println("\nResult of test case #2");
36             NotationTokenReader nreader = new NotationTokenReader();
37             nreader.parseDiceNotation("2d2-1");
38             Dice dice = new Dice(nreader);
39             CardStackRemovedCards csrc = new CardStackRemovedCards();
40
41             CardStack cardStack = new CardStack(dice, nreader, csrc);
42             System.out.println("diceNotation: " + cardStack.getDiceNotation());
43             System.out.println("Cards to remove: " + nreader.getToRemove());
44

```

```
45         } catch (Exception ex) {  
46             System.out.println(ex.getMessage());  
47         }  
48     }  
49 }  
50
```

Output



The screenshot shows an IDE output window with the following text:

```
run:  
Result of test case #1  
diceNotation: 2d2  
Cards to remove: 0  
  
Result of test case #2  
diceNotation: 2d2-1  
Cards to remove: 1  
BUILD SUCCESSFUL (total time: 0 seconds)
```

After constructor invoked populateCardStack & removeCard methods, it added shuffled cards to CardStack.

Since, there is no getter for CardStack's elements that test package can access, debug mode was used to test if populateCardStack was invoked by constructor with correct results. Following are the results:

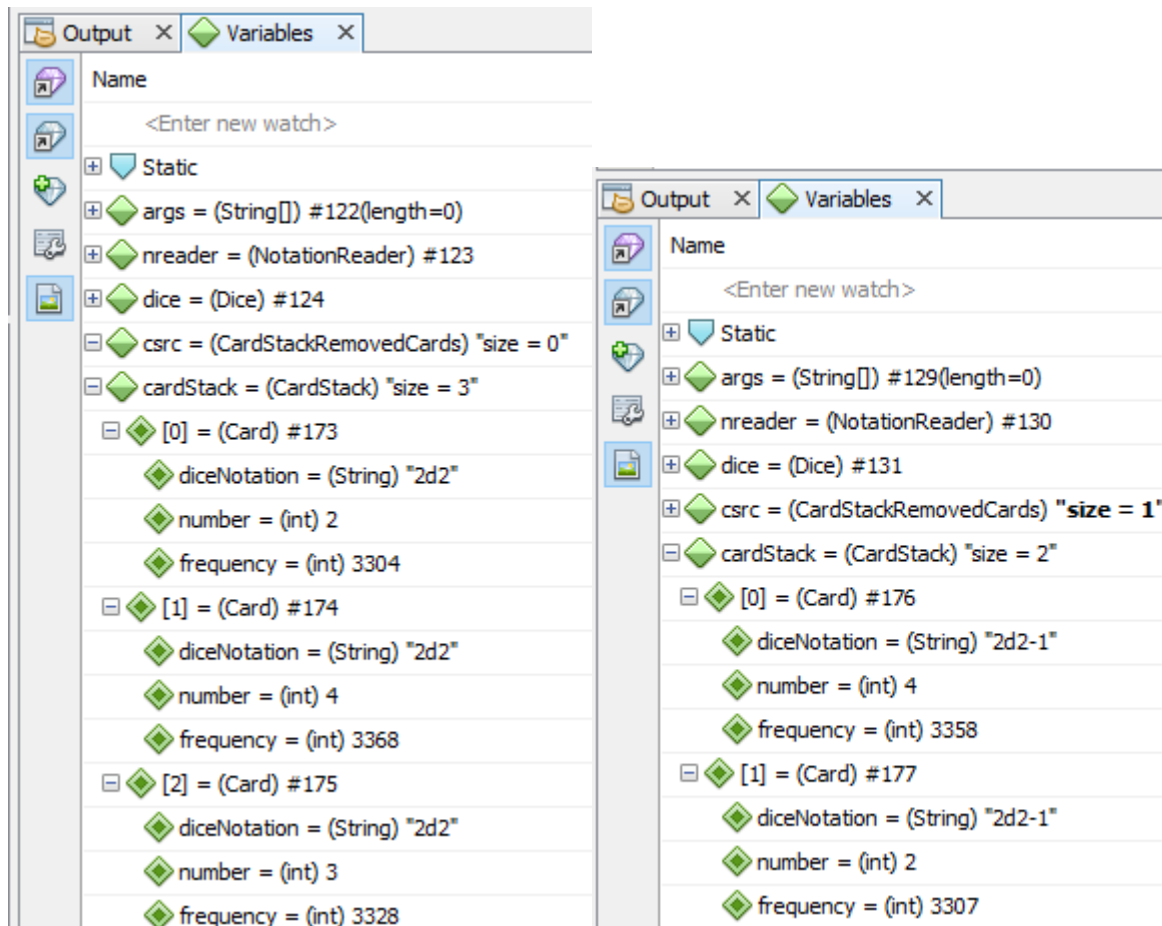


Figure: Checking if populateCardStack invoked by constructor removed correct no. of cards. For test case #1 (left) & test case#2.

Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

Test Suite No.9

Testingclass: cardstacks.CardStackRemovedCards

Testingtype: Black Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	ArrayList<Card> getRemovedCards(String diceNotation)	"3d2-4"	Valid string	Returns correct no. i.e. of removed cards. i.e. 4 cards.

Method of test data selection:

This method always receives valid input only, because it is only invoked after NotationReader class that was checked in Test Suite No.1 & 2, validates the input from GUI.

Test Execution

Source Code

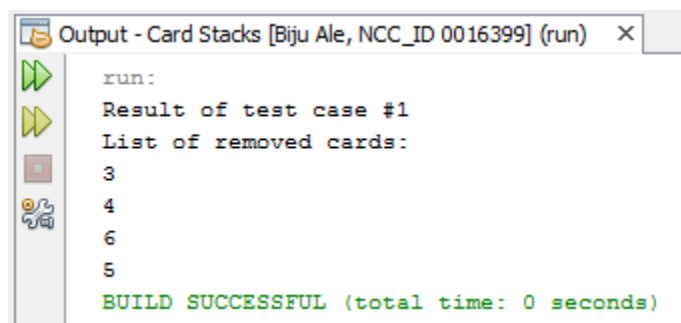
[PLEASE TURN OVER]

```

6   package test;
7
8   import cardstacks.Card;
9   import cardstacks.CardStack;
10  import cardstacks.CardStackRemovedCards;
11  import cardstacks.Dice;
12  import cardstacks.NotationReader;
13  import java.util.ArrayList;
14
15  /**
16   *
17   * @author Biju Ale
18   */
19  public class BlackBxTestCardStackRemovedCards {
20
21      public static void main(String[] args) {
22          try {
23              System.out.println("Result of test case #1");
24              NotationReader nreader = new NotationReader();
25              nreader.parseDiceNotation("3d2-4");
26              Dice dice = new Dice(nreader);
27              CardStackRemovedCards csrc = new CardStackRemovedCards();
28              CardStack cardStack = new CardStack(dice, nreader, csrc);
29
30              System.out.println("List of removed cards:");
31              ArrayList<Card> removedCards = csrc.getRemovedCards("3d2-4");
32              for (Card removedCard : removedCards) {
33                  System.out.println(removedCard.getNumber());
34              }
35
36          } catch (Exception ex) {
37              System.out.println(ex.getMessage());
38          }
39      }
40  }

```

Output



```

Output - Card Stacks [Biju Ale, NCC_ID 0016399] (run)
run:
Result of test case #1
List of removed cards:
3
4
6
5
BUILD SUCCESSFUL (total time: 0 seconds)

```

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	ArrayList<Card> getRemovedCards(String diceNotation)	"3d2- 4"	Valid string	Returns correct no. i.e. of removed cards. i.e. 4 cards.	Yes

Test Summary

From the above test results, all tests were executed as expected.

Test Suite No.10

Testingclass: cardstacks.CardStackRemovedCards

Testingtype: White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	ArrayList<Card> getRemovedCards(String diceNotation)	"3d2-4"	Valid string	Returns correct no. i.e. of removed cards. i.e. 4 removed cards.

Method of test data selection:

This method always receives valid input only, because it is only invoked after NotationReader class that was checked in Test Suite No.1 & 2, validates the input from GUI.

Test Execution

Source Code

[PLEASE TURN OVER]


```

6   package test;
7
8   import cardstacks.Card;
9   import cardstacks.CardStack;
10  import cardstacks.CardStackRemovedCards;
11  import cardstacks.Dice;
12  import cardstacks.NotationTokenReader;
13  import java.util.ArrayList;
14
15  /**
16   *
17   * @author Biju Ale
18   */
19  public class WhiteBxTestCardStackRemovedCards {
20      public static void main(String[] args) {
21          try {
22              System.out.println("Result of test case #1");
23              NotationTokenReader nreader = new NotationTokenReader();
24              nreader.parseDiceNotation("3d2-4");
25              Dice dice = new Dice(nreader);
26              CardStackRemovedCards csrc = new CardStackRemovedCards();
27              CardStack cardStack = new CardStack(dice, nreader, csrc);
28
29              System.out.println("List of removed cards:");
30              ArrayList<Card> removedCards = csrc.getRemovedCards("3d2-4");
31              for (Card removedCard : removedCards) {
32                  System.out.println(removedCard.getNumber());
33              }
34
35          } catch (Exception ex) {
36              System.out.println(ex.getMessage());
37          }
38      }
39  }

```

Output

```

Output - Card Stacks [Biju Ale, NCC_ID 0016399] (run) X
run:
Result of test case #1
List of removed cards:
3
4
6
5
BUILD SUCCESSFUL (total time: 0 seconds)

```

Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

Test Suite No. 11

Testingclass: cardstacks.HistoryDice

Testingtype: Black Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	boolean addToDiceHistory(Dice dice)	dice	Valid object	Returns TRUE after adding the dice to history.
2	boolean addToDiceHistory(Dice)	dice	Invalid object (duplicate dice)	Returns FALSE after matching dice to its history.

Method of test data selection:

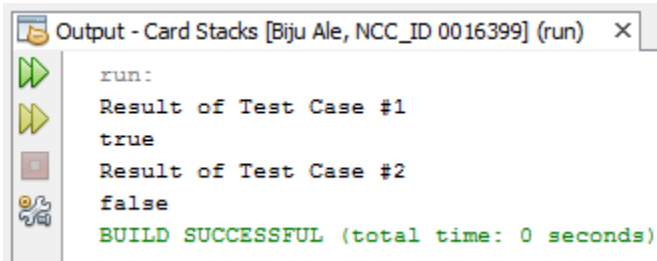
This method does not receive null, because it is only invoked after NotationReader class that was checked in Test Suite No.1 & 2, validates the null input from GUI.

Test Execution

Source Code

```
6 package test;
7
8 import cardstacks.Dice;
9 import cardstacks.HistoryDice;
10 import cardstacks.NotationReader;
11
12 /**
13  *
14  * @author Biju Ale
15  */
16 public class BlackBxTestHistoryDice {
17
18     public static void main(String[] args) {
19         System.out.println("Result of Test Case #1");
20         NotationReader nreader = new NotationReader();
21         try {
22             nreader.parseDiceNotation("4d4");
23             Dice dice = new Dice(nreader);
24             HistoryDice hd = new HistoryDice();
25             System.out.println(hd.addToDiceHistory(dice));
26
27             System.out.println("Result of Test Case #2");
28             System.out.println(hd.addToDiceHistory(dice));
29         } catch (Exception ex) {
30             System.out.println(ex.getMessage());
31         }
32     }
33 }
34 }
```


Output



```
run:
Result of Test Case #1
true
Result of Test Case #2
false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual Outcome as expected?
1	addToDiceHistory(Dice dice)	dice	Valid object	Returns TRUE after adding the dice to history.	Yes
2	addToDiceHistory(Dice)	dice	Invalid object (duplicate dice)	Returns FALSE after matching dice to its history.	Yes

Test Summary

From the above test results, all tests were executed as expected.

Test Suite No.12

Testingclass: cardstacks.HistoryDice

Testingtype: White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	boolean addToDiceHistory(Dice dice)	dice	Valid object	Returns TRUE after adding the dice to history.
2	boolean addToDiceHistory(Dice)	dice	Invalid object (duplicate dice)	Returns FALSE after matching dice to its history.

Method of test data selection:

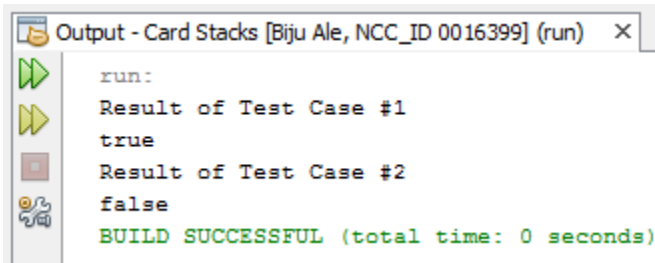
This method does not receive null input only, because it is only invoked after NotationReader class that was checked in Test Suite No.1 & 2, validates null input from GUI.

Test Execution

Source Code

```
6 package test;
7
8 import cardstacks.Dice;
9 import cardstacks.HistoryDice;
10 import cardstacks.NotationReader;
11
12 /**
13  *
14  * @author Biju Ale
15  */
16 public class WhiteBxTestHistoryDice {
17
18     public static void main(String[] args) {
19         System.out.println("Result of Test Case #1");
20         NotationReader nreader = new NotationReader();
21         try {
22             nreader.parseDiceNotation("4d4");
23             Dice dice = new Dice(nreader);
24             HistoryDice hd = new HistoryDice();
25             System.out.println(hd.addToDiceHistory(dice));
26
27             System.out.println("Result of Test Case #2");
28             System.out.println(hd.addToDiceHistory(dice));
29         } catch (Exception ex) {
30             System.out.println(ex.getMessage());
31         }
32     }
33 }
34 }
```

Output



```
run:
Result of Test Case #1
true
Result of Test Case #2
false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

Test Suite No. 13

Testingclass: cardstacks.CardStackDealtCards

Testingtype: Black Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	ArrayList<Card> getDealtCards(String diceNotation)	"4d4"	Valid string	Returns correct no. i.e. of dealt cards. i.e. 4 dealt cards.

Method of test data selection:

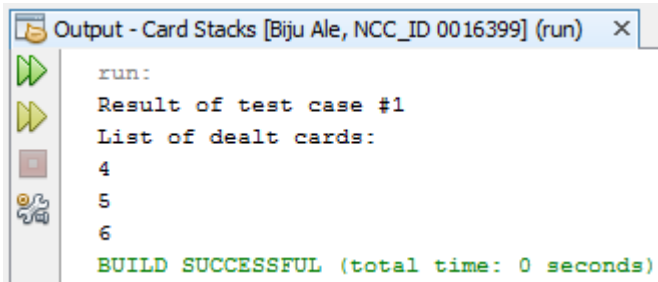
This method always receives valid input only, because it is only invoked after NotationReader class that was checked in Test Suite No.1 & 2, validates the input from GUI.

Test Execution

Source Code

```
6 package test;
7
8 import cardstacks.Card;
9 import cardstacks.CardStackDealtCards;
10 import java.util.ArrayList;
11
12 /**
13  *
14  * @author Biju Ale
15  */
16 public class BlackBxTestCardStackDealtCards {
17
18     public static void main(String[] args) {
19         try {
20             System.out.println("Result of test case #1");
21
22             CardStackDealtCards csdc = new CardStackDealtCards();
23             csdc.add(new Card(4, 455, "4d4"));
24             csdc.add(new Card(5, 345, "4d4"));
25             csdc.add(new Card(6, 453, "4d4"));
26
27             System.out.println("List of dealt cards:");
28             ArrayList<Card> dealtCards = csdc.getDealtCards("4d4");
29             for (Card dealtCard : dealtCards) {
30                 System.out.println(dealtCard.getNumber());
31             }
32
33         } catch (Exception ex) {
34             System.out.println(ex.getMessage());
35         }
36     }
37 }
```


Output



```
run:
Result of test case #1
List of dealt cards:
4
5
6
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	ArrayList<Card> getDealtCards(String diceNotation)	"4d4"	Valid string	Returns correct no. i.e. of dealt cards. i.e. 4 dealt cards.	Yes

Test Summary

From the above test results, all tests were executed as expected.

Test Suite No. 14

Testingclass: cardstacks.CardStackDealtCards

Testingtype: White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	ArrayList<Card> getDealtCards(String diceNotation)	"4d4"	Valid string	Returns correct no. i.e. of dealt cards. i.e. 4 dealt cards.

Method of test data selection:

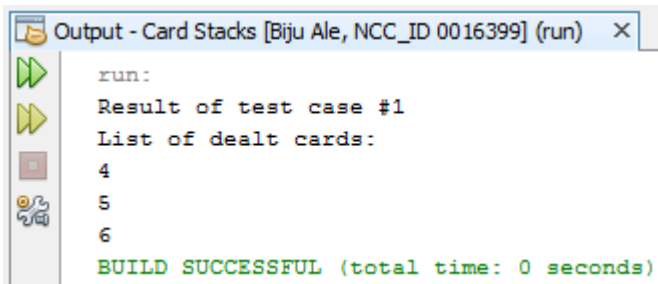
This method always receives valid input only, because it is only invoked after NotationReader class that was checked in Test Suite No.1 & 2, validates the input from GUI.

Test Execution

Source Code

```
6 package test;
7
8 import cardstacks.Card;
9 import cardstacks.CardStackDealtCards;
10 import java.util.ArrayList;
11
12 /**
13  *
14  * @author Biju Ale
15  */
16 public class WhiteBxTestCardStackDealtCards {
17
18     public static void main(String[] args) {
19         try {
20             System.out.println("Result of test case #1");
21
22             CardStackDealtCards csdc = new CardStackDealtCards();
23             csdc.add(new Card(4, 455, "4d4"));
24             csdc.add(new Card(5, 345, "4d4"));
25             csdc.add(new Card(6, 453, "4d4"));
26
27             System.out.println("List of dealt cards:");
28             ArrayList<Card> dealtCards = csdc.getDealtCards("4d4");
29             for (Card dealtCard : dealtCards) {
30                 System.out.println(dealtCard.getNumber());
31             }
32
33         } catch (Exception ex) {
34             System.out.println(ex.getMessage());
35         }
36     }
37 }
```

Output



```
run:
Result of test case #1
List of dealt cards:
4
5
6
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

Test Suite No.15

Testingclass: cardstacks.CollectionCardStacks

Testingtype: Black Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	Card moveDealtCard(String diceNotation, CardStackDealtCards csdc)	("2d2", csdc)	Valid objects	Returns dealt card by removing first card from the cardstack.

Method of test data selection:

This method always receives valid input only, because it is only invoked after NotationReader class that was checked in Test Suite No.1 & 2, validates the input from GUI.

Test Execution

Source Code

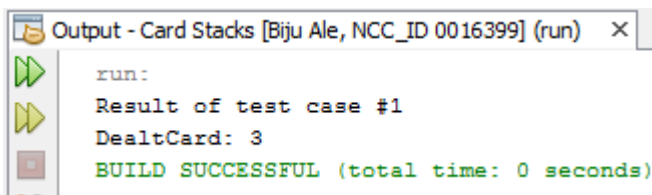
[PLEASE TURN OVER]

```

6 package test;
7
8 import cardstacks.Card;
9 import cardstacks.CardStack;
10 import cardstacks.CardStackDealtCards;
11 import cardstacks.CardStackRemovedCards;
12 import cardstacks.CollectionCardStacks;
13 import cardstacks.Dice;
14 import cardstacks.NotationTokenReader;
15
16 /**
17  *
18  * @author Biju Ale
19  */
20 public class BlackBxTestACollectionCardStacks {
21
22     public static void main(String[] args) {
23         System.out.println("Result of test case #1");
24         CollectionCardStacks ccs = new CollectionCardStacks();
25
26         CardStackRemovedCards csrc = new CardStackRemovedCards();
27         CardStackDealtCards csdc = new CardStackDealtCards();
28         NotationTokenReader nreader = new NotationTokenReader();
29         Dice dice;
30         try {
31             nreader.parseDiceNotation("2d2");
32             dice = new Dice(nreader);
33             CardStack cs = new CardStack(dice, nreader, csrc);
34             ccs.add(cs);
35
36             Card dealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
37             System.out.println("DealtCard: " + dealtCard.getNumber());
38
39         } catch (Exception ex) {
40             System.out.println(ex.getMessage());
41         }
42     }
43 }

```

Output



```

Output - Card Stacks [Biju Ale, NCC_ID 0016399] (run) X
run:
Result of test case #1
DealtCard: 3
BUILD SUCCESSFUL (total time: 0 seconds)

```

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	moveDealtCard(String diceNotation, CardStackDealtCards csdc)	("2d2", csdc)	Valid objects	Returns dealt card by removing first card from the cardstack.	Yes

Test Summary

From the above test results, all tests were executed as expected.

Test Suite No.16

Testingclass: cardstacks.CollectionCardStacks

Testingtype: White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	Card moveDealtCard(String diceNotation, CardStackDealtCards csdc)	("2d2", csdc)	Valid objects	Returns dealt card by removing first card from the cardstack.
2	Card moveDealtCard(String diceNotation, CardStackDealtCards csdc)	("2d2", csdc)	Valid objects	Repopulates stack when it is empty & returns dealt card by removing first card from the cardstack.

Method of test data selection:

- This method always receives valid input only, because it is only invoked after NotationReader class that was checked in Test Suite No.1 & 2, validates the input from GUI.
- Both test data are same, but in the second case, the test data is passed 3 times so that the 2d2 cardstack runs out of cards.

Test Execution

Source Code

[PLEASE TURN OVER]

```

6 package test;
7
8 import cardstacks.Card;
9 import cardstacks.CardStack;
10 import cardstacks.CardStackDealtCards;
11 import cardstacks.CardStackRemovedCards;
12 import cardstacks.CollectionCardStacks;
13 import cardstacks.Dice;
14 import cardstacks.NotationTokenReader;
15 import cardstacks.NotificationListener;
16
17 /**
18  *
19  * @author Biju Ale
20  */
21 public class WhiteBxTestACollectionCardStacks implements NotificationListener {
22
23     CollectionCardStacks ccs;
24     String notification;
25
26     WhiteBxTestACollectionCardStacks() {
27         ccs = new CollectionCardStacks();
28         ccs.addNotificationListener(this);
29
30         System.out.println("Result of test case #1");
31         CardStackRemovedCards csrc = new CardStackRemovedCards();
32         CardStackDealtCards csdc = new CardStackDealtCards();
33         NotationTokenReader nreader = new NotationTokenReader();
34         Dice dice;
35         try {
36             nreader.parseDiceNotation("2d2");
37             dice = new Dice(nreader);
38             CardStack cs = new CardStack(dice, nreader, csrc);
39             ccs.add(cs);
40             Card dealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
41             System.out.println("Dealt Card: " + dealtCard.getNumber());
42
43             System.out.println("\nResult of test case #1");

```

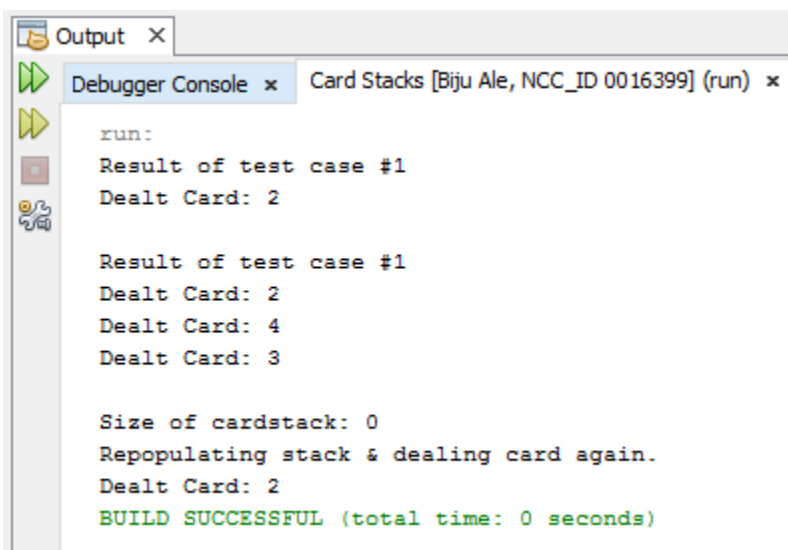


```

44         //1 card is already dealt above, now 2 cards are dealt so that the ↵
        carstack runs out of stack & returns dealtcard from repopulated stack.
45         System.out.println("Dealt Card: " + dealtCard.getNumber());
46         for (int i = 2; i < 4; i++) {
47             Card dealtCard2 = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
48             System.out.println("Dealt Card: " + dealtCard2.getNumber());
49         }
50
51         System.out.println("\nSize of cardstack: " + cs.size());
52         System.out.println("Repopulating stack & dealing card again.");
53         Card newDealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
54         System.out.println("Dealt Card: " + newDealtCard.getNumber());
55
56     } catch (Exception ex) {
57         System.out.println(ex.getMessage());
58         ex.printStackTrace();
59     }
60 }
61
62 public static void main(String[] args) {
63     new WhiteBoxTestACollectionCardStacks();
64 }
65
66 @Override
67 public void send(String notification) {
68     this.notification = notification;
69 }
70 }

```

Output



```

Output x
Debugger Console x Card Stacks [Biju Ale, NCC_ID 0016399] (run) x
run:
Result of test case #1
Dealt Card: 2

Result of test case #1
Dealt Card: 2
Dealt Card: 4
Dealt Card: 3

Size of cardstack: 0
Repopulating stack & dealing card again.
Dealt Card: 2
BUILD SUCCESSFUL (total time: 0 seconds)

```

Test Summary

While testing moveDealtCard method, implicitly rePopulateStack also got white-box & black-box tested.

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

Test Suite No.17

Testingclass: cardstacks.NotationTokenReader

Testingtype: Black Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	boolean shuffleStack(String diceNotation)	"4d4"	Valid string	Returns true after changing order of 4d4 cardstack.
2	boolean shuffleStack(String diceNotation)	"xxx"	Valid string	Returns false.
3	boolean shuffleStack(String diceNotation)	"	Null	Returns false.

Test Execution

Source Code

[PLEASE TURN OVER]

```

6 package test;
7
8 import cardstacks.CardStack;
9 import cardstacks.CardStackDealtCards;
10 import cardstacks.CardStackRemovedCards;
11 import cardstacks.CollectionCardStacks;
12 import cardstacks.Dice;
13 import cardstacks.NotationTokenReader;
14
15 /**
16  *
17  * @author Biju Ale
18  */
19 public class BlackBxTestBCollectionCardStacks {
20
21     public static void main(String[] args) {
22         CollectionCardStacks ccs = new CollectionCardStacks();
23
24         CardStackRemovedCards csrc = new CardStackRemovedCards();
25         CardStackDealtCards csdc = new CardStackDealtCards();
26         NotationTokenReader nreader = new NotationTokenReader();
27         Dice dice;
28         try {
29             nreader.parseDiceNotation("2d2");
30             dice = new Dice(nreader);
31             CardStack cs = new CardStack(dice, nreader, csrc);
32             ccs.add(cs);
33             System.out.println("CardStack 2d2 added. Only 2d2 exists in the collection↵
34             .");
35
36             System.out.println("\nResult of test case #1");
37             System.out.println("Input for shuffling: 2d2");
38             if (ccs.shuffleStack("2d2")) {
39                 System.out.println("TRUE");
40             }

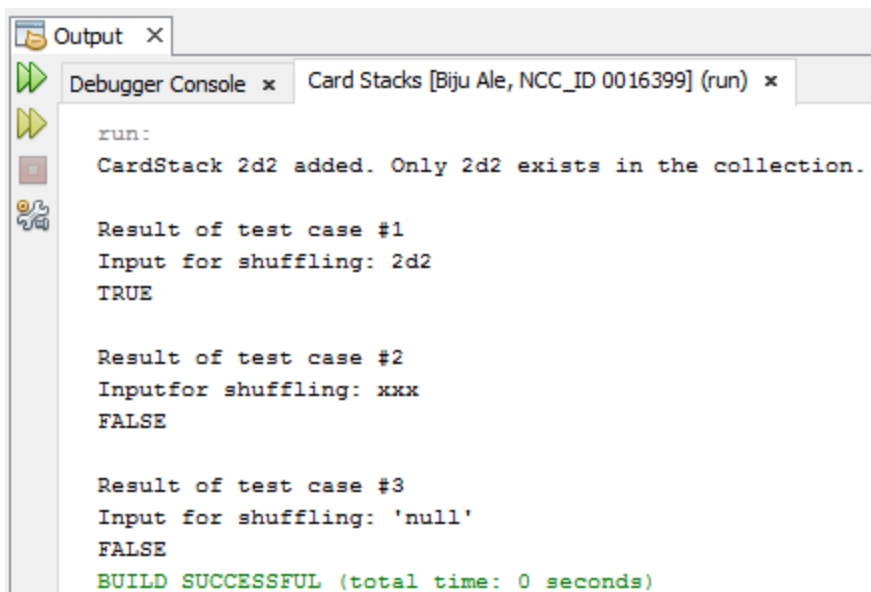
```

```

41         System.out.println("\nResult of test case #2");
42         System.out.println("Input for shuffling: xxx");
43         if (!ccs.shuffleStack("xxx")) {
44             System.out.println("FALSE");
45         }
46
47         System.out.println("\nResult of test case #3");
48         System.out.println("Input for shuffling: 'null'");
49         if (!ccs.shuffleStack("")) {
50             System.out.println("FALSE");
51         }
52
53     } catch (Exception ex) {
54         System.out.println(ex.getMessage());
55     }
56 }
57 }

```

Output



```

run:
CardStack 2d2 added. Only 2d2 exists in the collection.

Result of test case #1
Input for shuffling: 2d2
TRUE

Result of test case #2
Inputfor shuffling: xxx
FALSE

Result of test case #3
Input for shuffling: 'null'
FALSE
BUILD SUCCESSFUL (total time: 0 seconds)

```

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	boolean shuffleStack(String diceNotation)	"4d4"	Valid string	Returns true after changing order of 4d4 cardstack.	Yes
2	boolean shuffleStack(String diceNotation)	"xxx"	Valid string	Returns false.	Yes
3	boolean shuffleStack(String diceNotation)	""	Null	Returns false.	Yes

Test Summary

From the above test results, all tests were executed as expected.

Test Suite No.18

Testingclass: cardstacks.NotationsReader

Testingtype: White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	boolean shuffleStack(String diceNotation)	"4d4"	Valid string	Returns true after changing order of 4d4 cardstack.
2	boolean shuffleStack(String diceNotation)	"xxx"	Valid string	Returns false.
3	boolean shuffleStack(String diceNotation)	"	Null	Returns false.

Test Execution

Source Code

```

6 package test;
7
8 import cardstacks.CardStack;
9 import cardstacks.CardStackDealtCards;
10 import cardstacks.CardStackRemovedCards;
11 import cardstacks.CollectionCardStacks;
12 import cardstacks.Dice;
13 import cardstacks.NotationTokenReader;
14
15 /**
16  *
17  * @author Biju Ale
18  */
19 public class WhiteBxTestBCollectionCardStacks {
20
21     public static void main(String[] args) {
22         CollectionCardStacks ccs = new CollectionCardStacks();
23
24         CardStackRemovedCards csrc = new CardStackRemovedCards();
25         CardStackDealtCards csdc = new CardStackDealtCards();
26         NotationTokenReader nreader = new NotationTokenReader();
27         Dice dice;
28         try {
29             nreader.parseDiceNotation("2d2");
30             dice = new Dice(nreader);
31             CardStack cs = new CardStack(dice, nreader, csrc);
32             ccs.add(cs);
33             System.out.println("CardStack 2d2 added. Only 2d2 exists in the collection↵
34             .");
35
36             System.out.println("\nResult of test case #1");
37             System.out.println("Input for shuffling: 2d2");
38             if (ccs.shuffleStack("2d2")) {
39                 System.out.println("TRUE");
40             }
41
42             System.out.println("\nResult of test case #2");
43             System.out.println("Input for shuffling: xxx");

```

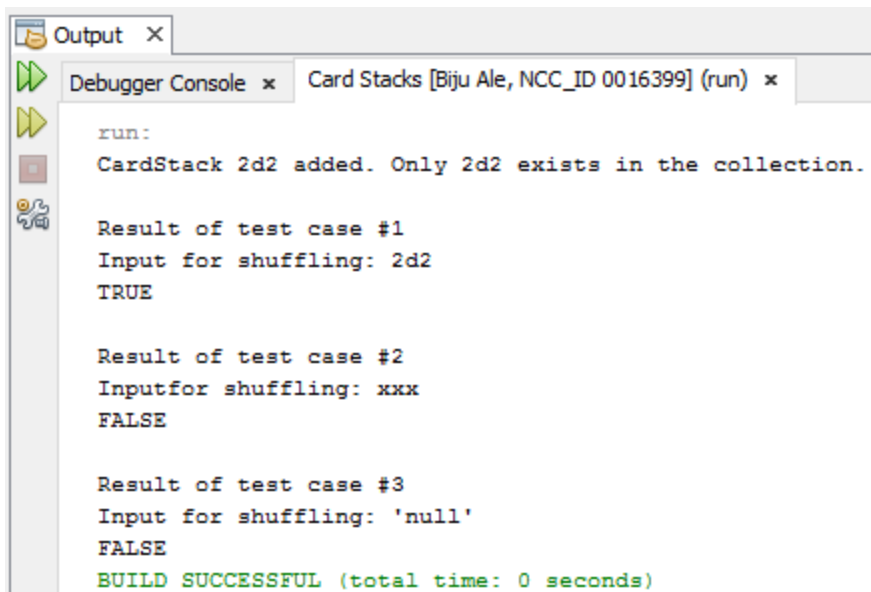


```

43         if (!ccs.shuffleStack("xxx")) {
44             System.out.println("FALSE");
45         }
46
47         System.out.println("\nResult of test case #3");
48         System.out.println("Input for shuffling: 'null'");
49         if (!ccs.shuffleStack("")) {
50             System.out.println("FALSE");
51         }
52
53     } catch (Exception ex) {
54         System.out.println(ex.getMessage());
55     }
56 }
57 }

```

Output



```

run:
CardStack 2d2 added. Only 2d2 exists in the collection.

Result of test case #1
Input for shuffling: 2d2
TRUE

Result of test case #2
Inputfor shuffling: xxx
FALSE

Result of test case #3
Input for shuffling: 'null'
FALSE

BUILD SUCCESSFUL (total time: 0 seconds)

```

Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

Test Suite No.19

Testingclass: cardstacks.NotationTokenReader

Testingtype: Black Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	CardStack getFutureCards(String diceNotation)	"2d2"	Valid string	Returns CardStack containing future cards.
2	CardStack getFutureCards(String diceNotation)	"xxx"	Invalid string	Returns null.
3	CardStack getFutureCards(String diceNotation)	null	Invalid string	Returns null.

Test Execution

Source Code

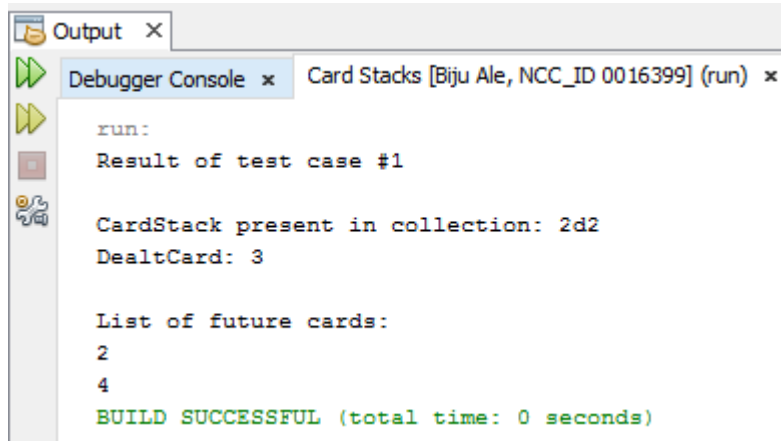
[PLEASE TURN OVER]

```

6 package test;
7
8 import cardstacks.Card;
9 import cardstacks.CardStack;
10 import cardstacks.CardStackDealtCards;
11 import cardstacks.CardStackRemovedCards;
12 import cardstacks.CollectionCardStacks;
13 import cardstacks.Dice;
14 import cardstacks.NotationTokenReader;
15
16 /**
17  *
18  * @author Biju Ale
19  */
20 public class BlackBxTestCCollectionCardStacks {
21
22     public static void main(String[] args) {
23         System.out.println("Result of test case #1");
24
25         CollectionCardStacks ccs = new CollectionCardStacks();
26         CardStackDealtCards csdc = new CardStackDealtCards();
27         CardStackRemovedCards csrc = new CardStackRemovedCards();
28         NotationTokenReader nreader = new NotationTokenReader();
29         Dice dice;
30
31         try {
32             nreader.parseDiceNotation("2d2");
33             dice = new Dice(nreader);
34             CardStack cs = new CardStack(dice, nreader, csrc);
35             ccs.add(cs);
36             System.out.println("\nCardStack present in collection: " + cs.getDiceNotation());
37             Card dealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
38             System.out.println("DealtCard: " + dealtCard.getNumber());
39
40             CardStack futureCards = ccs.getFutureCards(nreader.getDiceNotation());
41             System.out.println("\nList of future cards:");
42             for (Card futureCard : futureCards) {
43                 System.out.println(futureCard.getNumber());
44             }
45
46         } catch (Exception ex) {
47             System.out.println(ex.getMessage());
48         }
49     }
50 }

```

Output



The screenshot shows an IDE's Output window with a tab titled "Output x". Below it, a "Debugger Console x" tab is active, showing the output of a test run for "Card Stacks [Biju Ale, NCC_ID 0016399] (run)". The output text is as follows:

```
run:
Result of test case #1

CardStack present in collection: 2d2
DealtCard: 3

List of future cards:
2
4
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	CardStack getFutureCards(String diceNotation)	"2d2"	Valid string	Returns CardStack containing future cards.	Yes
2	CardStack getFutureCards(String diceNotation)	"xxx"	Invalid string	Returns null.	Yes
3	CardStack getFutureCards(String diceNotation)	null	Invalid string	Returns null.	Yes

Test Summary

From the above test results, all tests were executed as expected.

Test Suite No.20

Testingclass: cardstacks.NotationTokenReader

Testingtype: White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	CardStack getFutureCards(String diceNotation)	"2d2"	Valid string	Returns CardStack containing future cards.
2	CardStack getFutureCards(String diceNotation)	"xxx"	Invalid string	Returns null.
3	CardStack getFutureCards(String diceNotation)	null	Invalid string	Returns null.

Test Execution

Source Code

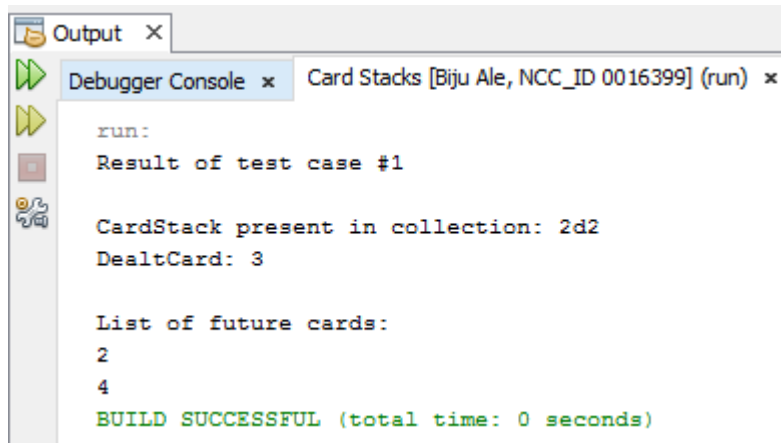
[PLEASE TURN OVER]

```

6   package test;
7
8   import cardstacks.Card;
9   import cardstacks.CardStack;
10  import cardstacks.CardStackDealtCards;
11  import cardstacks.CardStackRemovedCards;
12  import cardstacks.CollectionCardStacks;
13  import cardstacks.Dice;
14  import cardstacks.NotationTokenReader;
15
16  /**
17   *
18   * @author Biju Ale
19   */
20  public class WhiteBxTestCCollectionCardStacks {
21
22      public static void main(String[] args) {
23          System.out.println("Result of test case #1");
24
25          CollectionCardStacks ccs = new CollectionCardStacks();
26          CardStackDealtCards csdc = new CardStackDealtCards();
27          CardStackRemovedCards csrc = new CardStackRemovedCards();
28          NotationTokenReader nreader = new NotationTokenReader();
29          Dice dice;
30
31          try {
32              nreader.parseDiceNotation("2d2");
33              dice = new Dice(nreader);
34              CardStack cs = new CardStack(dice, nreader, csrc);
35              ccs.add(cs);
36              System.out.println("\nCardStack present in collection: " + cs.getDiceNotation());
37              Card dealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
38              System.out.println("DealtCard: " + dealtCard.getNumber());
39
40              CardStack futureCards = ccs.getFutureCards(nreader.getDiceNotation());
41              System.out.println("\nList of future cards:");
42              for (Card futureCard : futureCards) {
43                  System.out.println(futureCard.getNumber());
44              }
45
46          } catch (Exception ex) {
47              System.out.println(ex.getMessage());
48          }
49      }
50  }
51

```

Output



```
run:
Result of test case #1

CardStack present in collection: 2d2
DealtCard: 3

List of future cards:
2
4
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

Test Suite No.21

Testingclass: cardstacks.Canvas

Testingtype: Black Box & White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	void sendSingleDealtCard(Card singleDealtCard)	singleDealtCard	Valid object	Draws single dealt card on JPanel
2	void sendSingleDealtCard(Card singleDealtCard)	singleDealtCard	Null (object has null data member i.e. null dice notation)	Throws exception with message – “INVALID DICE NOTATION”
3	void sendSingleDealtCard(Card singleDealtCard)	singleDealtCard	Null (object has invalid data member i.e. invalid dice notation)	Throws exception with message – “INVALID DICE NOTATION”

Test Execution

Source Code


```

6   package test;
7
8   import cardstacks.Canvas;
9   import cardstacks.Card;
10  import cardstacks.CardStack;
11  import cardstacks.CardStackDealtCards;
12  import cardstacks.CardStackRemovedCards;
13  import cardstacks.CollectionCardStacks;
14  import cardstacks.Dice;
15  import cardstacks.DrawActionListener;
16  import cardstacks.NotationalReader;
17  import cardstacks.NotificationListener;
18  import java.awt.Graphics;
19  import javax.swing.JFrame;
20
21  /**
22   *
23   * @author Biju Ale
24   */
25  public class BlackWhiteBxTestACanvas extends JFrame implements
NotificationListener {
26
27      static Canvas canvas;
28      String notification;
29      Graphics g;
30      static DrawActionListener drawActionListener;
31
32  void testA() { //For Test case #1
33      canvas = new Canvas();
34      drawActionListener = canvas;
35      canvas.addNotificationListener(this);
36
37      CollectionCardStacks ccs = new CollectionCardStacks();
38      CardStackRemovedCards csrc = new CardStackRemovedCards();
39      CardStackDealtCards csdc = new CardStackDealtCards();
40      NotationalReader nreader = new NotationalReader();
41      Dice dice;
42
43      try {
44          nreader.parseDiceNotation("2d2");
45          dice = new Dice(nreader);

```

```

45         dice = new Dice(nreader);
46         CardStack cs = new CardStack(dice, nreader, csrc);
47         ccs.add(cs);
48         Card singlDealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), ←
csdc);
49
50         canvas.sendSingleDealtCard(singlDealtCard);
51         canvas.ACTION_DRAW = Canvas.DRAW_FOR_GET_CARD;
52
53     } catch (Exception ex) {
54         System.out.println(ex.getMessage());
55     }
56 }
57
58 void testB() { //For Test case #2
59     System.out.println("\nResult of test case #2");
60     canvas = new Canvas();
61     drawActionListener = canvas;
62     canvas.addNotificationListener(this);
63
64     CollectionCardStacks ccs = new CollectionCardStacks();
65     CardStackRemovedCards csrc = new CardStackRemovedCards();
66     CardStackDealtCards csdc = new CardStackDealtCards();
67     NotationReader nreader = new NotationReader();
68     Dice dice;
69
70     try {
71         nreader.parseDiceNotation("xxx");
72         dice = new Dice(nreader);
73         CardStack cs = new CardStack(dice, nreader, csrc);
74         ccs.add(cs);
75         Card singlDealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), ←
csdc);
76
77         canvas.sendSingleDealtCard(singlDealtCard);
78         canvas.ACTION_DRAW = Canvas.DRAW_FOR_GET_CARD;
79
80     } catch (Exception ex) {
81         System.out.println(ex.getMessage());
82     }
83 }

```

```

84
85 void testC() { //For Test case #3
86     System.out.println("\nResult of test case #3");
87     canvas = new Canvas();
88     drawActionListener = canvas;
89     canvas.addNotificationListener(this);
90
91     CollectionCardStacks ccs = new CollectionCardStacks();
92     CardStackRemovedCards csrc = new CardStackRemovedCards();
93     CardStackDealtCards csdc = new CardStackDealtCards();
94     NotationReader nreader = new NotationReader();
95     Dice dice;
96
97     try {
98         nreader.parseDiceNotation("");
99         dice = new Dice(nreader);
100         CardStack cs = new CardStack(dice, nreader, csrc);
101         ccs.add(cs);
102         Card singlDealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), ↵
csdc);
103
104         canvas.sendSingleDealtCard(singlDealtCard);
105         canvas.ACTION_DRAW = Canvas.DRAW_FOR_GET_CARD;
106
107     } catch (Exception ex) {
108         System.out.println(ex.getMessage());
109     }
110 }
111
112 public static void main(String[] args) {
113     BlackWhiteBxTestACanvas btc = new BlackWhiteBxTestACanvas();
114     btc.setSize(500, 500);
115     btc.setVisible(true);
116
117     btc.testA();
118     btc.add(canvas);
119
120     btc.testB();
121
122     btc.testC();
123
124 }
125
126 @Override
127 public void send(String notification) {
128     this.notification = notification;
129 }
130 }
131

```

Output

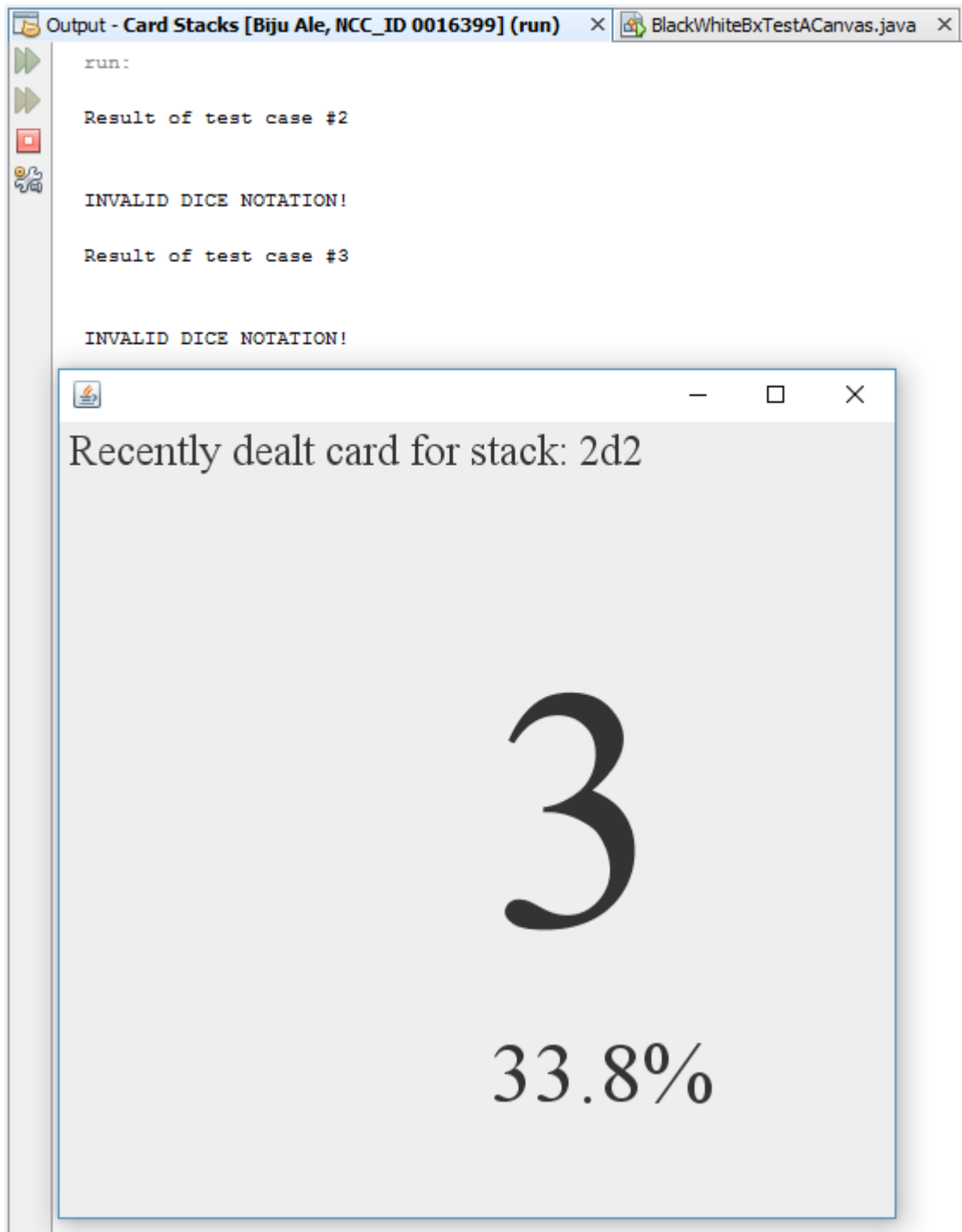


Figure: The output window shows result for test case #2 and test case #3 whereas the JFrame shows the result of test case #1

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	void sendSingleDealtCard(Card singleDealtCard)	singleDealtCard	Valid object	Draws single dealt card on JPanel	Yes
2	void sendSingleDealtCard(Card singleDealtCard)	singleDealtCard	Null (object has null data member i.e. null dice notation)	Throws exception with message – “INVALID DICE NOTATION”	Yes
3	void sendSingleDealtCard(Card singleDealtCard)	singleDealtCard	Null (object has invalid data member i.e. invalid dice notation)	Throws exception with message – “INVALID DICE NOTATION”	Yes

Test Summary

While black box test was performed, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution. Therefore, it is safe to say that both black box and white box was performed simultaneously.

From the above test results, all tests were executed as expected.

Test Suite No.22

Testingclass: cardstacks.Canvas

Testingtype: Black Box & White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	sendAllDealtCards(ArrayList allDealtCards)	allDealtCards	Valid object	Draws Graph of all dealt cards on JPanel
2	sendAllDealtCards(ArrayList allDealtCards)	allDealtCards	Null (object has null data member i.e. null dice notation)	Throws exception with message – “INVALID DICE NOTATION”
3	sendAllDealtCards(ArrayList allDealtCards)	allDealtCards	Null (object has invalid data member i.e. invalid dice notation)	Throws exception with message – “INVALID DICE NOTATION”

Test Execution

Source Code

```

6      package test;
7
8      import cardstacks.Canvas;
9      import cardstacks.Card;
10     import cardstacks.CardStack;
11     import cardstacks.CardStackDealtCards;
12     import cardstacks.CardStackRemovedCards;
13     import cardstacks.CollectionCardStacks;
14     import cardstacks.Dice;
15     import cardstacks.DrawActionListener;
16     import cardstacks.NototationReader;
17     import cardstacks.NotificationListener;
18     import java.awt.Graphics;
19     import java.util.ArrayList;
20     import javax.swing.JFrame;
21
22     /**
23      *
24      * @author Biju Ale
25      */
26     public class BlackWhiteBxTestBCanvas extends JFrame implements NotificationListener {
27
28         static Canvas canvas;
29         String notification;
30         Graphics g;
31         static DrawActionListener drawActionListener;
32
33         void testA() { //For Test case #1
34             canvas = new Canvas();
35             drawActionListener = canvas;
36             canvas.addNotificationListener(this);
37
38             CollectionCardStacks ccs = new CollectionCardStacks();
39             CardStackRemovedCards csrc = new CardStackRemovedCards();
40             CardStackDealtCards csdc = new CardStackDealtCards();
41             NototationReader nreader = new NototationReader();
42             Dice dice;

```

```

43
44
45     try {
46         nreader.parseDiceNotation("4d4");
47         dice = new Dice(nreader);
48         CardStack cs = new CardStack(dice, nreader, csrc);
49         ccs.add(cs);
50
51         //Simulating multiple card dealing
52         ArrayList<Card> allDealtCards = new ArrayList();
53         for (int i = 0; i < 4; i++) {
54             allDealtCards.add(ccs.moveDealtCard(nreader.getDiceNotation(), csdc));
55         }
56         canvas.sendAllDealtCards(allDealtCards);
57         canvas.ACTION_DRAW = Canvas.DRAW_FOR_DEALT_CARD;
58     } catch (Exception ex) {
59         System.out.println(ex.getMessage());
60     }
61
62 void testB() { //For Test case #2
63     System.out.println("\nResult of test case #2");
64     canvas = new Canvas();
65     drawActionListener = canvas;
66     canvas.addNotificationListener(this);
67
68     CollectionCardStacks ccs = new CollectionCardStacks();
69     CardStackRemovedCards csrc = new CardStackRemovedCards();
70     CardStackDealtCards csdc = new CardStackDealtCards();
71     NotationReader nreader = new NotationReader();
72     Dice dice;
73
74     try {
75         nreader.parseDiceNotation("xxx");
76         dice = new Dice(nreader);
77         CardStack cs = new CardStack(dice, nreader, csrc);
78         ccs.add(cs);
79
80         //Simulating multiple card dealing
81         ArrayList<Card> allDealtCards = new ArrayList();
82         for (int i = 0; i < 4; i++) {

```



```

83         allDealtCards.add(ccs.moveDealtCard(nreader.getDiceNotation(), csdc));
84     }
85     canvas.sendAllDealtCards(allDealtCards);
86     canvas.ACTION_DRAW = Canvas.DRAW_FOR_DEALT_CARD;
87
88     } catch (Exception ex) {
89         System.out.println(ex.getMessage());
90     }
91 }
92
93 void testC() { //For Test case #3
94     System.out.println("\nResult of test case #3");
95     canvas = new Canvas();
96     drawActionListener = canvas;
97     canvas.addNotificationListener(this);
98
99     CollectionCardStacks ccs = new CollectionCardStacks();
100     CardStackRemovedCards csrc = new CardStackRemovedCards();
101     CardStackDealtCards csdc = new CardStackDealtCards();
102     NotationReader nreader = new NotationReader();
103     Dice dice;
104
105     try {
106         nreader.parseDiceNotation("");
107         dice = new Dice(nreader);
108         CardStack cs = new CardStack(dice, nreader, csrc);
109         ccs.add(cs);
110
111         //Simulating multiple card dealing
112         ArrayList<Card> allDealtCards = new ArrayList();
113         for (int i = 0; i < 4; i++) {
114             allDealtCards.add(ccs.moveDealtCard(nreader.getDiceNotation(), csdc));
115         }
116         canvas.sendAllDealtCards(allDealtCards);
117         canvas.ACTION_DRAW = Canvas.DRAW_FOR_DEALT_CARD;
118
119     } catch (Exception ex) {
120         System.out.println(ex.getMessage());
121     }
122 }
123

```

```
124  public static void main(String[] args) {
125      BlackWhiteBxTestBCanvas btc = new BlackWhiteBxTestBCanvas();
126      btc.setSize(500, 500);
127      btc.setVisible(true);
128
129      btc.testA();
130      btc.add(canvas);
131
132      btc.testB();
133
134      btc.testC();
135
136  }
137
138  @Override
139  public void send(String notification) {
140      this.notification = notification;
141  }
142  }
```

Output

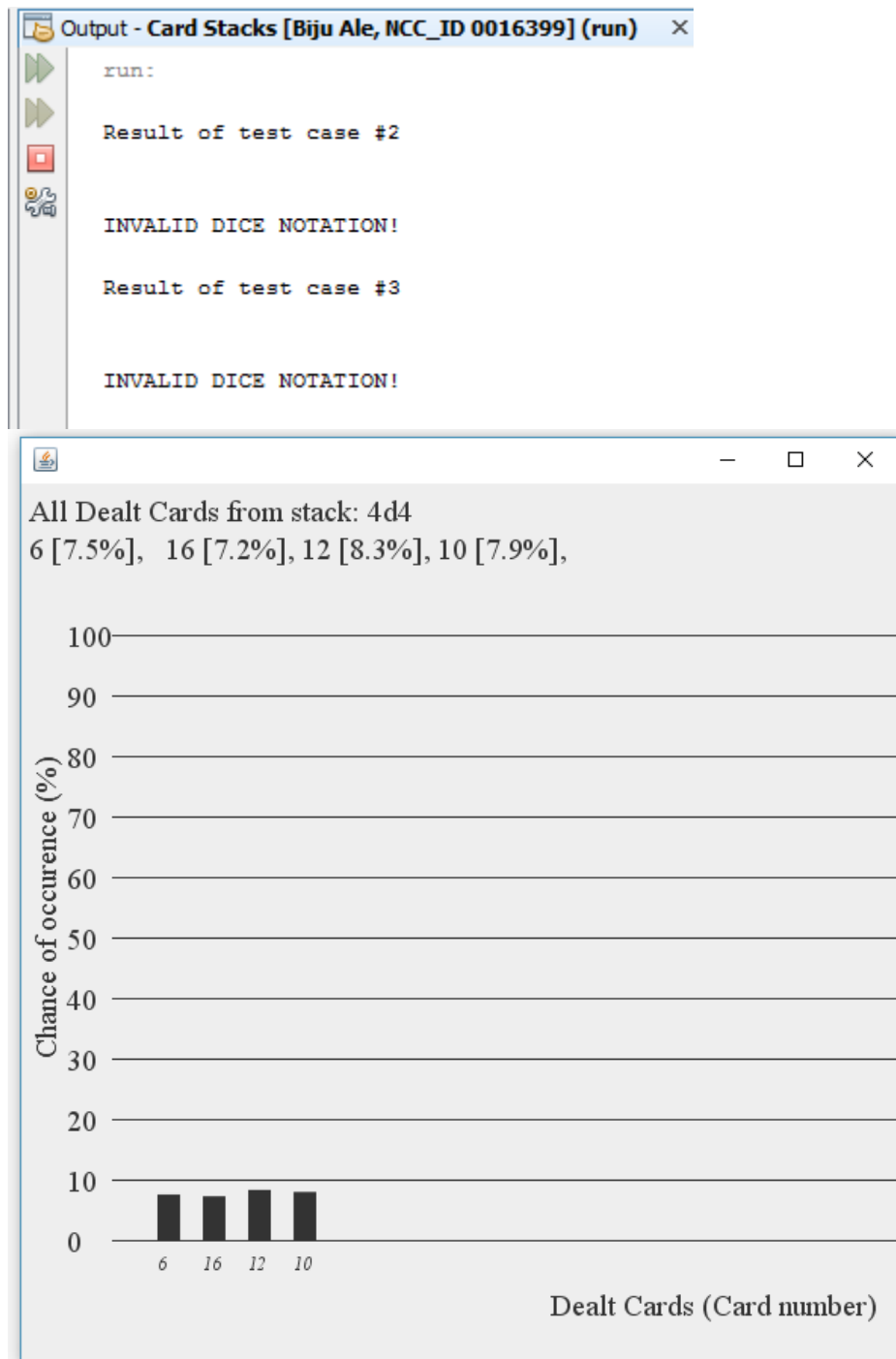


Figure: The output window shows result for test case #2 and test case #3 whereas the JFrame shows the result of test case #1

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	sendAllDealtCards(ArrayList allDealtCards)	allDealtCards	Valid object	Draws Graph of all dealt cards on JPanel	Yes
2	sendAllDealtCards(ArrayList allDealtCards)	allDealtCards	Null (object has null data member i.e. null dice notation)	Throws exception with message – "INVALID DICE NOTATION"	Yes
3	sendAllDealtCards(ArrayList allDealtCards)	allDealtCards	Null (object has invalid data member i.e. invalid dice notation)	Throws exception with message – "INVALID DICE NOTATION"	Yes

Test Summary

While black box test was performed, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution. Therefore, it is safe to say that both black box and white box was performed simultaneously.

From the above test results, all tests were executed as expected.

Test Suite No.23

Testingclass: cardstacks.Canvas

Testingtype: Black Box & White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	sendRemovedCards(ArrayList allRemovedCards)	allRemovedCards	Valid object	Draws all removed cards on JPanel
2	sendRemovedCards(ArrayList allRemovedCards)	allRemovedCards	Null (object has null data member i.e. null dice notation)	Throws exception with message – “INVALID DICE NOTATION”
3	sendRemovedCards(ArrayList allRemovedCards)	allRemovedCards	Null (object has invalid data member i.e. invalid dice notation)	Throws exception with message – “INVALID DICE NOTATION”

Test Execution

Source Code

```

6      package test;
7
8      import cardstacks.Canvas;
9      import cardstacks.Card;
10     import cardstacks.CardStack;
11     import cardstacks.CardStackDealtCards;
12     import cardstacks.CardStackRemovedCards;
13     import cardstacks.CollectionCardStacks;
14     import cardstacks.Dice;
15     import cardstacks.DrawActionListener;
16     import cardstacks.NototationReader;
17     import cardstacks.NotificationListener;
18     import java.awt.Graphics;
19     import java.util.ArrayList;
20     import javax.swing.JFrame;
21
22     /**
23      *
24      * @author Biju Ale
25      */
26     public class BlackWhiteBxTestCCanvas extends JFrame implements NotificationListener {
27
28         static Canvas canvas;
29         String notification;
30         Graphics g;
31         static DrawActionListener drawActionListener;
32
33         void testA() { //For Test case #1
34             canvas = new Canvas();
35             drawActionListener = canvas;
36             canvas.addNotificationListener(this);
37
38             CollectionCardStacks ccs = new CollectionCardStacks();
39             CardStackRemovedCards csrc = new CardStackRemovedCards();
40             CardStackDealtCards csdc = new CardStackDealtCards();
41             NototationReader nreader = new NototationReader();
42             Dice dice;
43
44             try {
45                 nreader.parseDiceNotation("4d4-4");
46                 dice = new Dice(nreader);

```

```

47         CardStack cs = new CardStack(dice, nreader, csrc);
48         ccs.add(cs);
49
50         csrc.getRemovedCards(nreader.getDiceNotation());
51         canvas.sendRemovedCards(csrc.getRemovedCards(nreader.getDiceNotation()));
52         canvas.ACTION_DRAW = Canvas.DRAW_FOR_REMOVED;
53     } catch (Exception ex) {
54         System.out.println(ex.getMessage());
55     }
56 }
57
58 void testB() { //For Test case #2
59     System.out.println("\nResult of test case #2");
60     canvas = new Canvas();
61     drawActionListener = canvas;
62     canvas.addNotificationListener(this);
63
64     CollectionCardStacks ccs = new CollectionCardStacks();
65     CardStackRemovedCards csrc = new CardStackRemovedCards();
66     CardStackDealtCards csdc = new CardStackDealtCards();
67     NotationReader nreader = new NotationReader();
68     Dice dice;
69
70     try {
71         nreader.parseDiceNotation("xxx");
72         dice = new Dice(nreader);
73         CardStack cs = new CardStack(dice, nreader, csrc);
74         ccs.add(cs);
75
76         csrc.getRemovedCards(nreader.getDiceNotation());
77         canvas.sendRemovedCards(csrc.getRemovedCards(nreader.getDiceNotation()));
78         canvas.ACTION_DRAW = Canvas.DRAW_FOR_REMOVED;
79
80     } catch (Exception ex) {
81         System.out.println(ex.getMessage());
82     }
83 }
84

```

```

85 void testC() { //For Test case #3
86     System.out.println("\nResult of test case #3");
87     canvas = new Canvas();
88     drawActionListener = canvas;
89     canvas.addNotificationListener(this);
90
91     CollectionCardStacks ccs = new CollectionCardStacks();
92     CardStackRemovedCards csrc = new CardStackRemovedCards();
93     CardStackDealtCards csdc = new CardStackDealtCards();
94     NotationReader nreader = new NotationReader();
95     Dice dice;
96
97     try {
98         nreader.parseDiceNotation("");
99         dice = new Dice(nreader);
100         CardStack cs = new CardStack(dice, nreader, csrc);
101         ccs.add(cs);
102
103         csrc.getRemovedCards(nreader.getDiceNotation());
104         canvas.sendRemovedCards(csrc.getRemovedCards(nreader.getDiceNotation()));
105         canvas.ACTION_DRAW = Canvas.DRAW_FOR_REMOVED;
106
107     } catch (Exception ex) {
108         System.out.println(ex.getMessage());
109     }
110 }
111
112 public static void main(String[] args) {
113     BlackWhiteBxTestCCanvas btc = new BlackWhiteBxTestCCanvas();
114     btc.setSize(500, 500);
115     btc.setVisible(true);
116
117     btc.testA();
118     btc.add(canvas);
119
120     btc.testB();
121
122     btc.testC();
123 }

```


Output

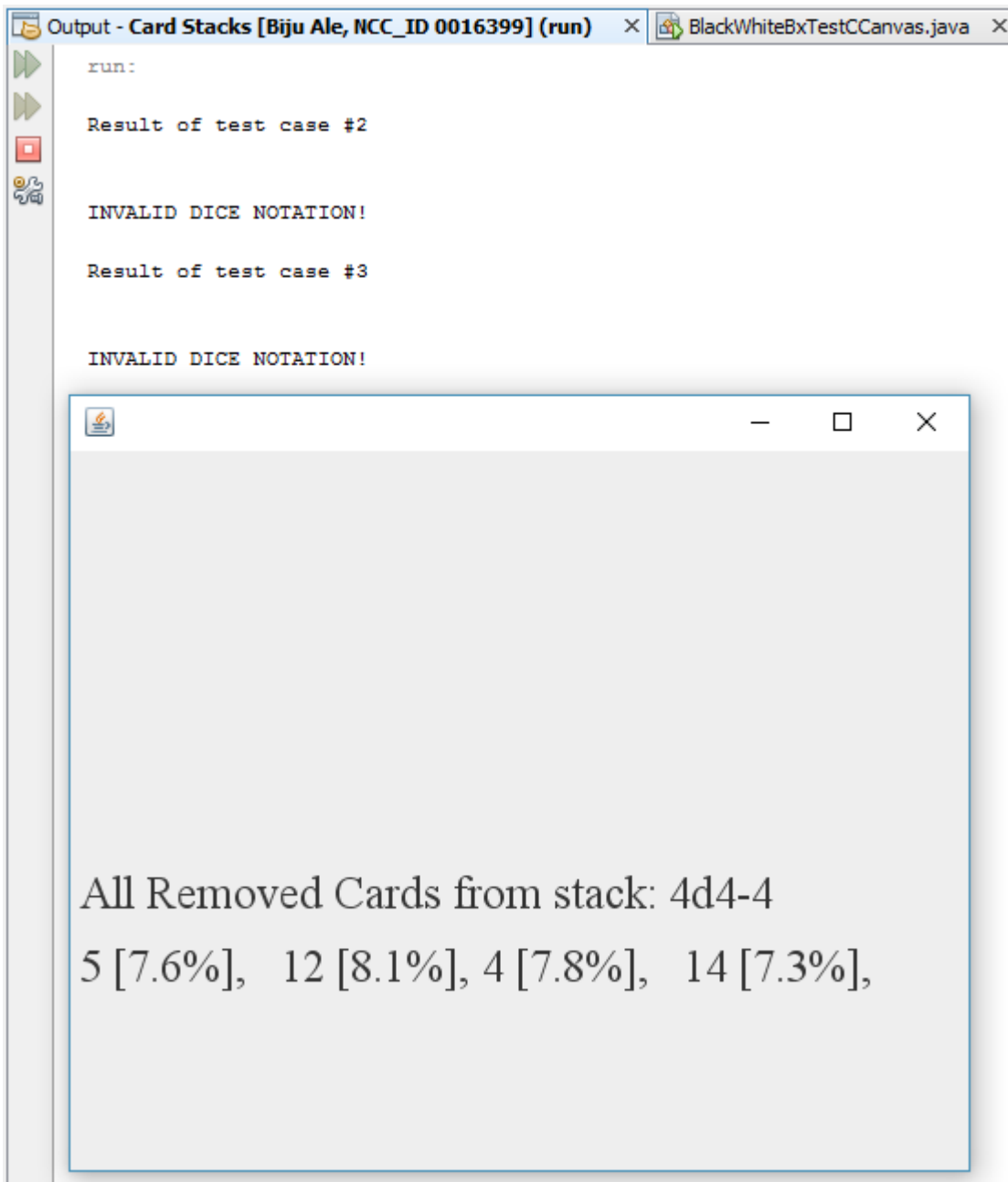


Figure: The output window shows result for test case #2 and test case #3 whereas the JFrame shows the result of test case #1

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	sendRemovedCards(ArrayList allRemovedCards)	allRemovedCards	Valid object	Draws all removed cards on JPanel	Yes
2	sendRemovedCards(ArrayList allRemovedCards)	allRemovedCards	Null (object has null data member i.e. null dice notation)	Throws exception with message – “INVALID DICE NOTATION”	Yes
3	sendRemovedCards(ArrayList allRemovedCards)	allRemovedCards	Null (object has invalid data member i.e. invalid dice notation)	Throws exception with message – “INVALID DICE NOTATION”	Yes

Test Summary

While black box test was performed, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution. Therefore, it is safe to say that both black box and white box was performed simultaneously.

From the above test results, all tests were executed as expected.

Test Suite No.24

Testingclass: cardstacks.Canvas

Testingtype: Black Box & White Box / Unit Testing

#	Method	Test Data	Input type	Expected Outcome
1	sendFutureCards(CardStack futureCards)	futureCards	Valid object	Draws graph of all future cards on JPanel
2	sendFutureCards(CardStack futureCards)	futureCards	Null (object has null data member i.e. null dice notation)	Throws exception with message – “INVALID DICE NOTATION”
3	sendFutureCards(CardStack futureCards)	futureCards	Invalid (object has invalid data member i.e. invalid dice notation)	Throws exception with message – “INVALID DICE NOTATION”

Test Execution

Source Code

```

6     package test;
7
8     import cardstacks.Canvas;
9     import cardstacks.CardStack;
10    import cardstacks.CardStackDealtCards;
11    import cardstacks.CardStackRemovedCards;
12    import cardstacks.CollectionCardStacks;
13    import cardstacks.Dice;
14    import cardstacks.DrawActionListener;
15    import cardstacks.NototationReader;
16    import cardstacks.NotificationListener;
17    import java.awt.Graphics;
18    import javax.swing.JFrame;
19
20    /**
21     *
22     * @author Biju Ale
23     */
24    public class BlackWhiteBxTestDCanvas extends JFrame implements NotificationListener {
25
26        static Canvas canvas;
27        String notification;
28        Graphics g;
29        static DrawActionListener drawActionListener;
30
31        void testA() { //For Test case #1
32            canvas = new Canvas();
33            drawActionListener = canvas;
34            canvas.addNotificationListener(this);
35
36            CollectionCardStacks ccs = new CollectionCardStacks();
37            CardStackRemovedCards csrc = new CardStackRemovedCards();
38            CardStackDealtCards csdc = new CardStackDealtCards();
39            NototationReader nreader = new NototationReader();
40            Dice dice;
41
42            try {
43                nreader.parseDiceNotation("2d2");
44                dice = new Dice(nreader);
45                CardStack cs = new CardStack(dice, nreader, csrc);

```

```

46         ccs.add(cs);
47
48         ccs.getFutureCards(nreader.getDiceNotation());
49         canvas.sendFutureCards(ccs.getFutureCards(nreader.getDiceNotation()));
50         canvas.ACTION_DRAW = Canvas.DRAW_FOR_FUTURE;
51     } catch (Exception ex) {
52         System.out.println(ex.getMessage());
53     }
54 }
55
56 void testB() { //For Test case #2
57     System.out.println("\nResult of test case #2");
58     canvas = new Canvas();
59     drawActionListener = canvas;
60     canvas.addNotificationListener(this);
61
62     CollectionCardStacks ccs = new CollectionCardStacks();
63     CardStackRemovedCards csrc = new CardStackRemovedCards();
64     CardStackDealtCards csdc = new CardStackDealtCards();
65     NotationReader nreader = new NotationReader();
66     Dice dice;
67
68     try {
69         nreader.parseDiceNotation("xxx");
70         dice = new Dice(nreader);
71         CardStack cs = new CardStack(dice, nreader, csrc);
72         ccs.add(cs);
73
74         ccs.getFutureCards(nreader.getDiceNotation());
75         canvas.sendFutureCards(ccs.getFutureCards(nreader.getDiceNotation()));
76         canvas.ACTION_DRAW = Canvas.DRAW_FOR_FUTURE;
77
78     } catch (Exception ex) {
79         System.out.println(ex.getMessage());
80     }
81 }
82
83 void testC() { //For Test case #3
84     System.out.println("\nResult of test case #3");
85     canvas = new Canvas();

```

```

86 drawActionListener = canvas;
87 canvas.addNotificationListener(this);
88
89 CollectionCardStacks ccs = new CollectionCardStacks();
90 CardStackRemovedCards csrc = new CardStackRemovedCards();
91 CardStackDealtCards csdc = new CardStackDealtCards();
92 NotationReader nreader = new NotationReader();
93 Dice dice;
94
95 try {
96     nreader.parseDiceNotation("");
97     dice = new Dice(nreader);
98     CardStack cs = new CardStack(dice, nreader, csrc);
99     ccs.add(cs);
100
101     ccs.getFutureCards(nreader.getDiceNotation());
102     canvas.sendFutureCards(ccs.getFutureCards(nreader.getDiceNotation()));
103     canvas.ACTION_DRAW = Canvas.DRAW_FOR_FUTURE;
104
105 } catch (Exception ex) {
106     System.out.println(ex.getMessage());
107 }
108
109
110 public static void main(String[] args) {
111     BlackWhiteBxTestDCanvas btc = new BlackWhiteBxTestDCanvas();
112     btc.setSize(500, 500);
113     btc.setVisible(true);
114
115     btc.testA();
116     btc.add(canvas);
117
118     btc.testB();
119
120     btc.testC();
121
122 }
123
124 @Override
125 public void send(String notification) {
126     this.notification = notification;
127 }
128
129

```

Output

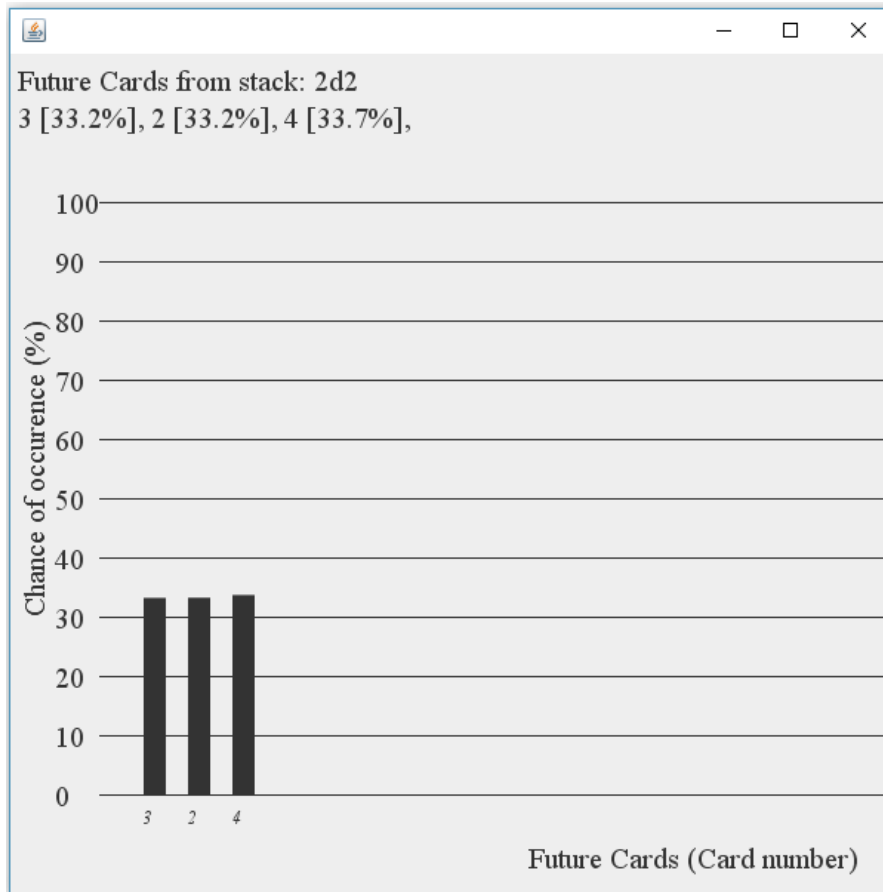
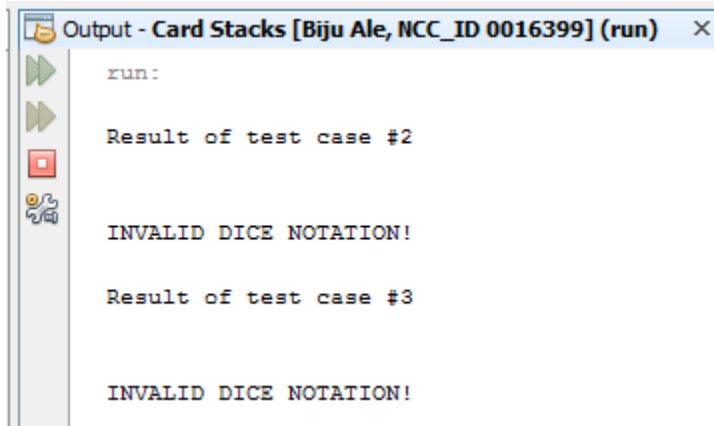


Figure: The output window shows result for test case #2 and test case #3 whereas the JFrame shows the result of test case #1

Test Result

#	Method	Test Data	Input type	Expected Outcome	Actual outcome as expected?
1	sendFutureCards(CardStack futureCards)	futureCards	Valid object	Draws graph of all future cards on JPanel	Yes
2	sendFutureCards(CardStack futureCards)	futureCards	Null (object has null data member i.e. null dice notation)	Throws exception with message – "INVALID DICE NOTATION"	Yes
3	sendFutureCards(CardStack futureCards)	futureCards	Invalid (object has invalid data member i.e. invalid dice notation)	Throws exception with message – "INVALID DICE NOTATION"	Yes

Test Summary

While black box test was performed, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution. Therefore, it is safe to say that both black box and white box was performed simultaneously.

From the above test results, all tests were executed as expected.

Final Test Suite No.25

Testingclass: cardstacks.NotationReader

Testingtype: Integration Testing

Test Suite No. 1 to 24 was designed to unit test all the methods of classes that make up the application. Now, integration testing is required to test the communication between objects. Main functionalities are shown executed as a final application, below.

Test Execution

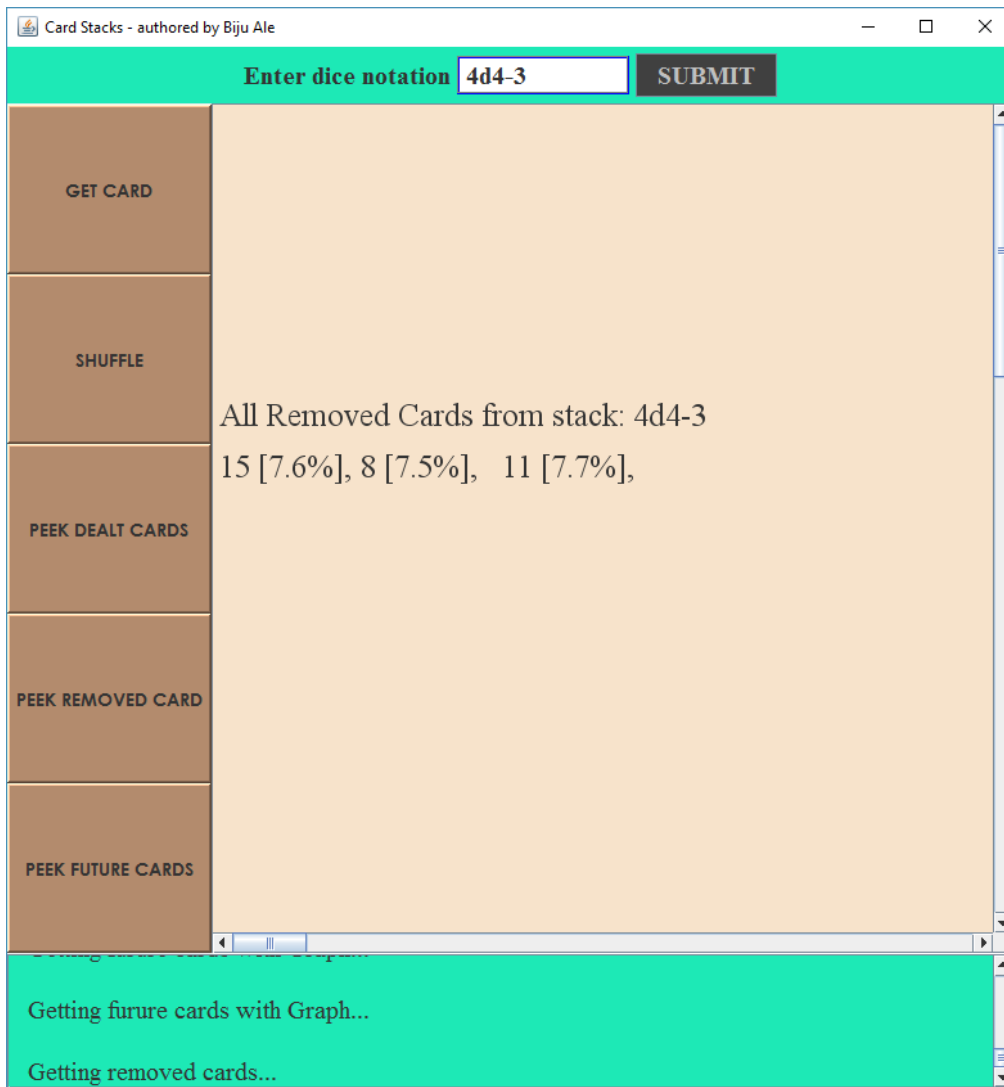
Please turn over....



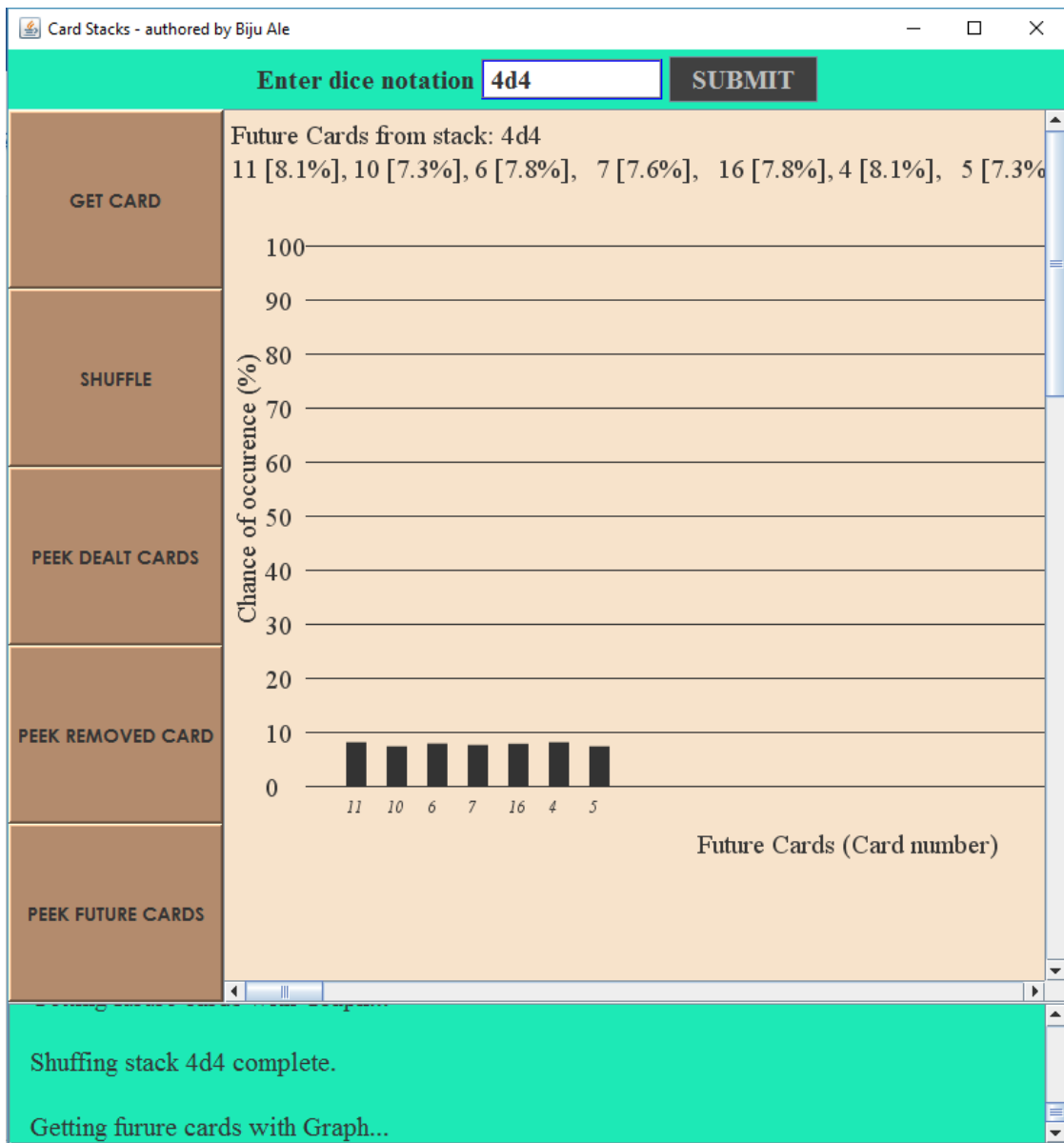
As shown in the above figure, invalid input dice notation triggered exception handling which threw the correct message.



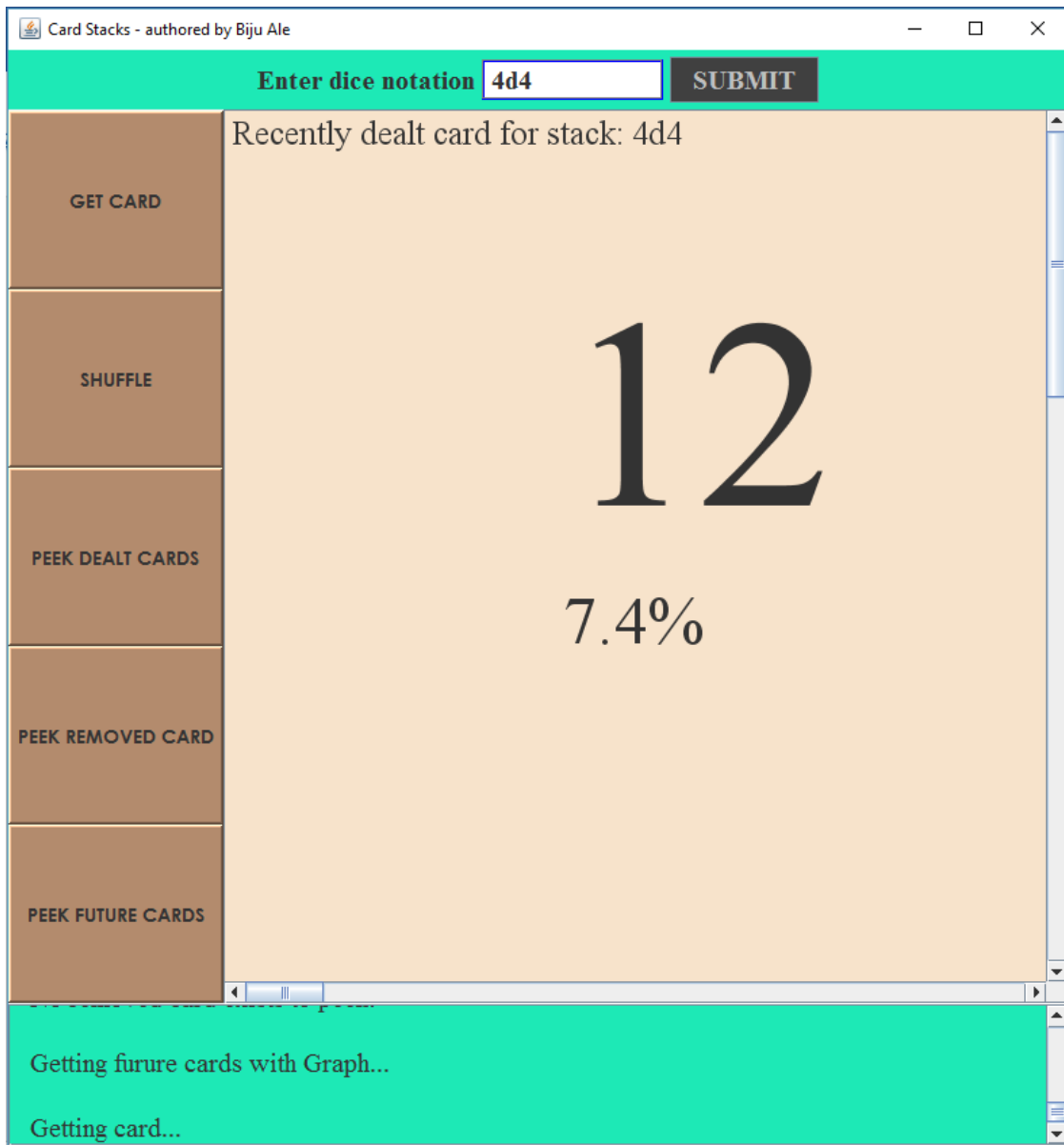
As shown in the above figure, null input for dice notation triggered exception handling which threw the correct message.



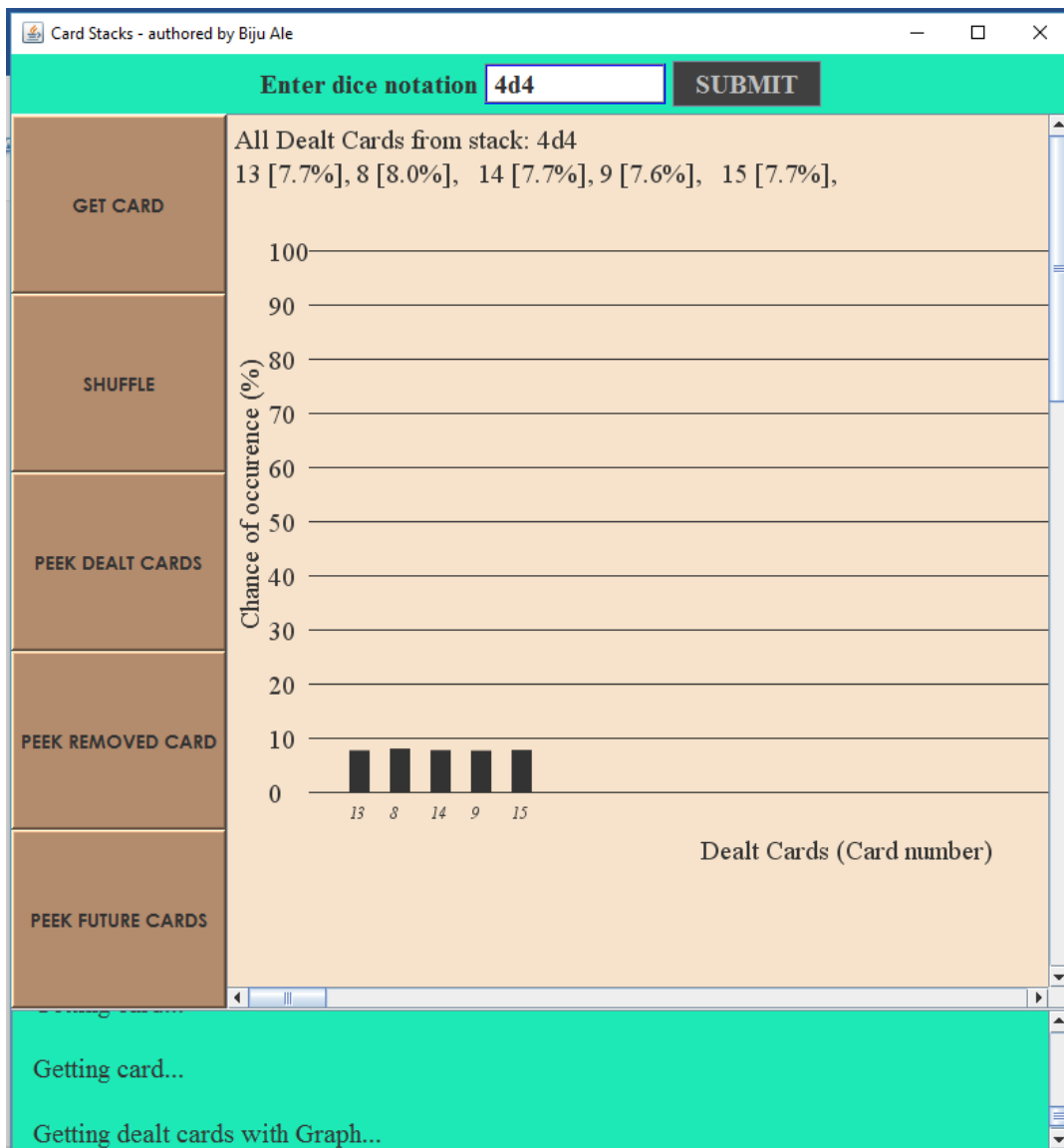
As shown in the above figure, application generated the correct number of cards to remove. Parsing of dual delimiter input – “d” and “-” was correctly executed.



As shown in the above figure, graph of future cards is correctly generated.



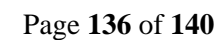
As shown in the above figure, a card is immediately dealt off once valid dice notation is passed.



As shown in the above figure, graph of dealt cards were correctly generated.

Test Summary

From the above test results, all tests were executed as expected. It is safe to say that the communication between objects has been validated. Integration testing was successfully accomplished without unexpected results.



Conclusion

In a nutshell, the report has incorporated 3 aspects of application documentation - the source code of the application, the testing carried out full coverage of verification & validation of the application and finally the UML based class diagram to provide the design overview.

In case of technical implementations, the application has provided the ability to generate combinations from multi-dimensional dices, by simulating the virtual dice. The integrity of the dice notation was maintained by validating user input. Exceptions were handled where relevant in order to make the application robust enough against unexpected runtime scenarios. Cohesion was maintained by distributing workload over several dedicated classes. Tight coupling was loosened using interfaces in case of notification passing & canvas paintings for graph.

Similarly, in case of testing of application. Each methods of all classes were black-box and white-box tested using relevant equivalence partitioning for test data. The tests were distributed across test suites for common type of test & the method being tested. Lastly, integration testing was carried out to check proper functioning of communication between the objects.

The report is a documented evidence for demonstrating the requirements fulfillment as mandated by the board game company. The power of object-oriented programming style was harnessed to deliver a reliable solution.

Referencing

Bell, D. & Parr, M., 2010. Exceptions. In: *Java for Students, 6th Ed.*. London: Pearson Education Limited, pp. 301-314.

Deitel, P. & Deitel, H., 2015. Regular Expressions, Class Patterns and Class Matcher. In: *Java How To Program, 10th Ed.*. New Jersey: Pearson, pp. 624-633.

Schildt, H., 2014. Painting in Swing. In: *Java The Complete Reference, 9th Edition*. New York: McGraw-Hill Education, pp. 1036-1040.

Vermeulen, A. et al., 2000. Documentation Conventions. In: *The Element of Java Style*. Cambridge: Cambridge University Press, pp. 31-52.

Page left intentionally blank.

Page left intentionally blank.