Table of Contents

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.

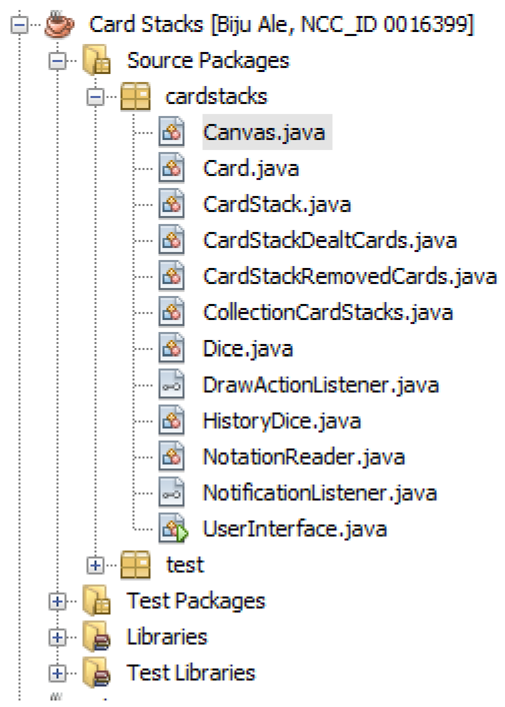
The java based application presented in this documentation, is specifically, and designed to solve the issue with regular dice rolls. In addition to storing the combination in the form of a card, it also promotes a fair randomness by optimizing the occurrence of each combination towards statistically 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)

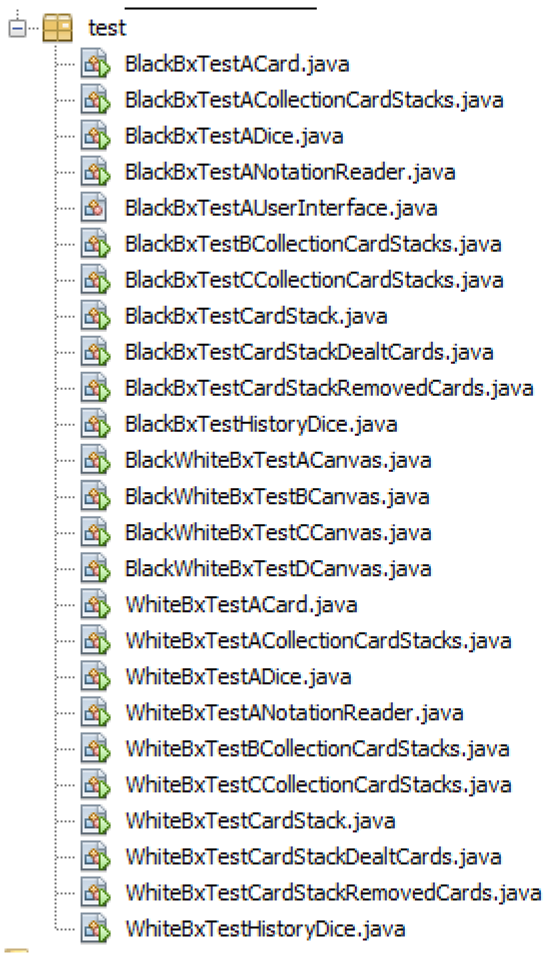
Overview of all Project Files

Main package files



```
Card Stacks [Biju Ale, NCC_ID 0016399]
    Source Packages
        cardstacks
            Canvas.java
            Card.java
            CardStack.java
            CardStackDealtCards.java
            CardStackRemovedCards.java
            CollectionCardStacks.java
            Dice.java
            DrawActionListener.java
            HistoryDice.java
            NotationReader.java
            NotificationListener.java
            UserInterface.java
        test
    Test Packages
    Libraries
    Test Libraries
```

Test package files



test
- BlackBxTestACard.java
- BlackBxTestACollectionCardStacks.java
- BlackBxTestADice.java
- BlackBxTestANotationReader.java
- BlackBxTestAUserInterface.java
- BlackBxTestBCollectionCardStacks.java
- BlackBxTestCCollectionCardStacks.java
- BlackBxTestCardStack.java
- BlackBxTestCardStackDealtCards.java
- BlackBxTestCardStackRemovedCards.java
- BlackBxTestHistoryDice.java
- BlackWhiteBxTestACanvas.java
- BlackWhiteBxTestBCanvas.java
- BlackWhiteBxTestCCanvas.java
- BlackWhiteBxTestDCanvas.java
- WhiteBxTestACard.java
- WhiteBxTestACollectionCardStacks.java
- WhiteBxTestADice.java
- WhiteBxTestANotationReader.java
- WhiteBxTestBCollectionCardStacks.java
- WhiteBxTestCCollectionCardStacks.java
- WhiteBxTestCardStack.java
- WhiteBxTestCardStackDealtCards.java
- WhiteBxTestCardStackRemovedCards.java
- WhiteBxTestHistoryDice.java

Class: NotationReader

[PLEASE TURN OVER]

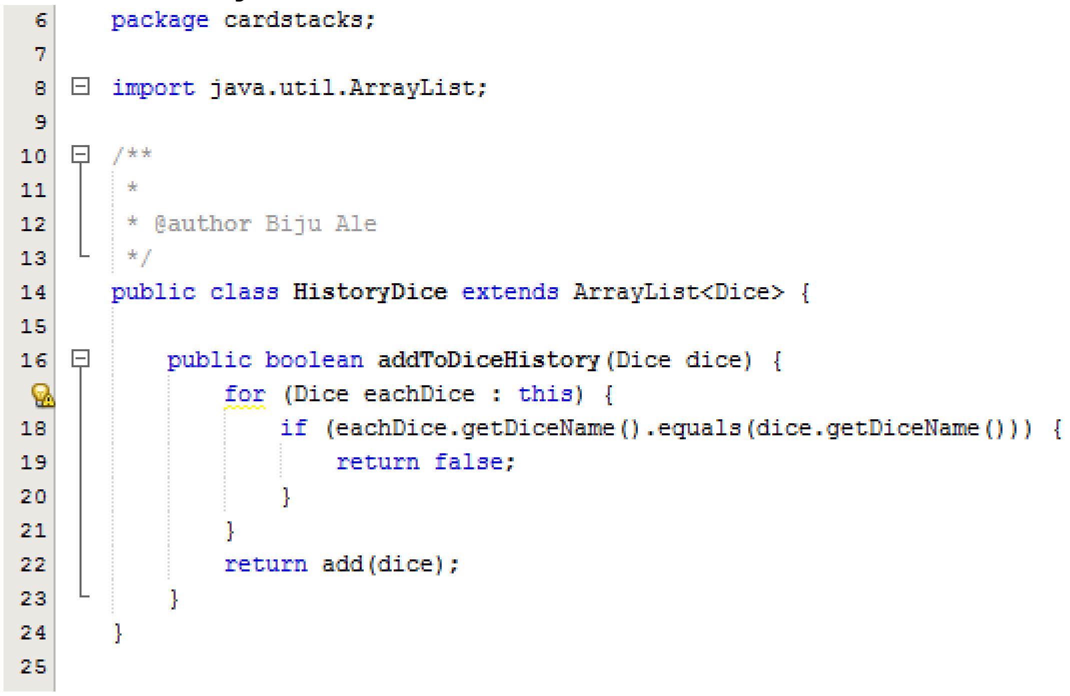
```java
package cardstacks;

import java.util.regex.Pattern;

/**
 *
 * @author Biju Ale
 */
public class NotationReader {

    private int numDices, numFaces, toRemove;
    private String diceNotation;

    public int getNumDices() {
        return numDices;
    }

    public int getNumFaces() {
        return numFaces;
    }

    public int getToRemove() {
        return toRemove;
    }

    public String getDiceNotation() {
        return diceNotation;
    }

    public void parseDiceNotation(String diceNotation) throws Exception {
        String[] parts;
        //Main part in Regex Pattern: (1|[1-9][0-9]*) [Matches any number without leading zeros.]
        if (Pattern.matches("((1|[1-9][0-9]*)d(1|[1-9][0-9]*))|((1|[1-9][0-9]*)d(1|[1-9][0-9]*)[-](1|[1-9][0-9]*))", diceNotation)) {
            parts = diceNotation.split("d|-");
            this.diceNotation = diceNotation;
            this.numDices = Integer.parseInt(parts[0]);
            this.numFaces = Integer.parseInt(parts[1]);
            this.toRemove = (diceNotation.contains("-")) ? Integer.parseInt(parts[2]) : 0; //Ternary Operator
        } else {
            throw new Exception("\n\nINVALID DICE NOTATION!");
        }
    }
}
```

Class: Dice

[PLEASE TURN OVER]

```java
package cardstacks;

import java.util.Arrays;
import java.util.Random;

/**
 *
 * @author Biju Ale
 */
public class Dice {

    private String diceName;
    public static final int ROLL_TIMES = 10000;
    private Integer[] Combinations;
    private Integer[] frequencies;
    private int minCombination, maxCombination;

    public Integer[] getCombinations() {
        return Combinations;
    }

    public Integer[] getFrequencies() {
        return frequencies;
    }

    public String getDiceName() {
        return diceName;
    }

    public int getMinCombination() {
        return minCombination;
    }

    public int getMaxCombination() {
        return maxCombination;
    }

    public Dice(NotationReader nreader) throws Exception {
        if (nreader.getToRemove() > (nreader.getNumDices() * nreader.getNumFaces())) {
            throw new Exception("No. of cards to remove cannot exceed total no. of cards. Enter valid notation.\n");
        }
        this.diceName = nreader.getDiceNotation();
        setMinMax(nreader.getNumDices(), nreader.getNumDices() * nreader.getNumFaces());
    }

    //set - min & max Combination
    private  void setMinMax(int minCombination, int maxCombination) {
        this.minCombination = minCombination;
        this.maxCombination = maxCombination;
        populateCombinations();
    }

    //Populate all possible combinations in Combination array
    private  void populateCombinations() {
        Combinations = new Integer[maxCombination - minCombination + 1];
        int index = 0;
        for (int eachCombination = minCombination; eachCombination < maxCombination + 1; eachCombination++) {
            Combinations[index] = eachCombination;
            index++;
        }
        roll(Combinations);
    }

    //Roll the dice & record combinations' frequecnies
    private  void roll(Integer[] Combinations) {
        int randomIndex;

        frequencies = new Integer[Combinations.length];

        Arrays.fill(frequencies, 0);//Reset all indexes
        //Save index frequencies
        Random rdmGenerator = new Random();
        for (int i = 0; i < ROLL_TIMES; i++) {
            randomIndex = rdmGenerator.nextInt(Combinations.length);
            frequencies[randomIndex] += 1;
        }
    }
}
```

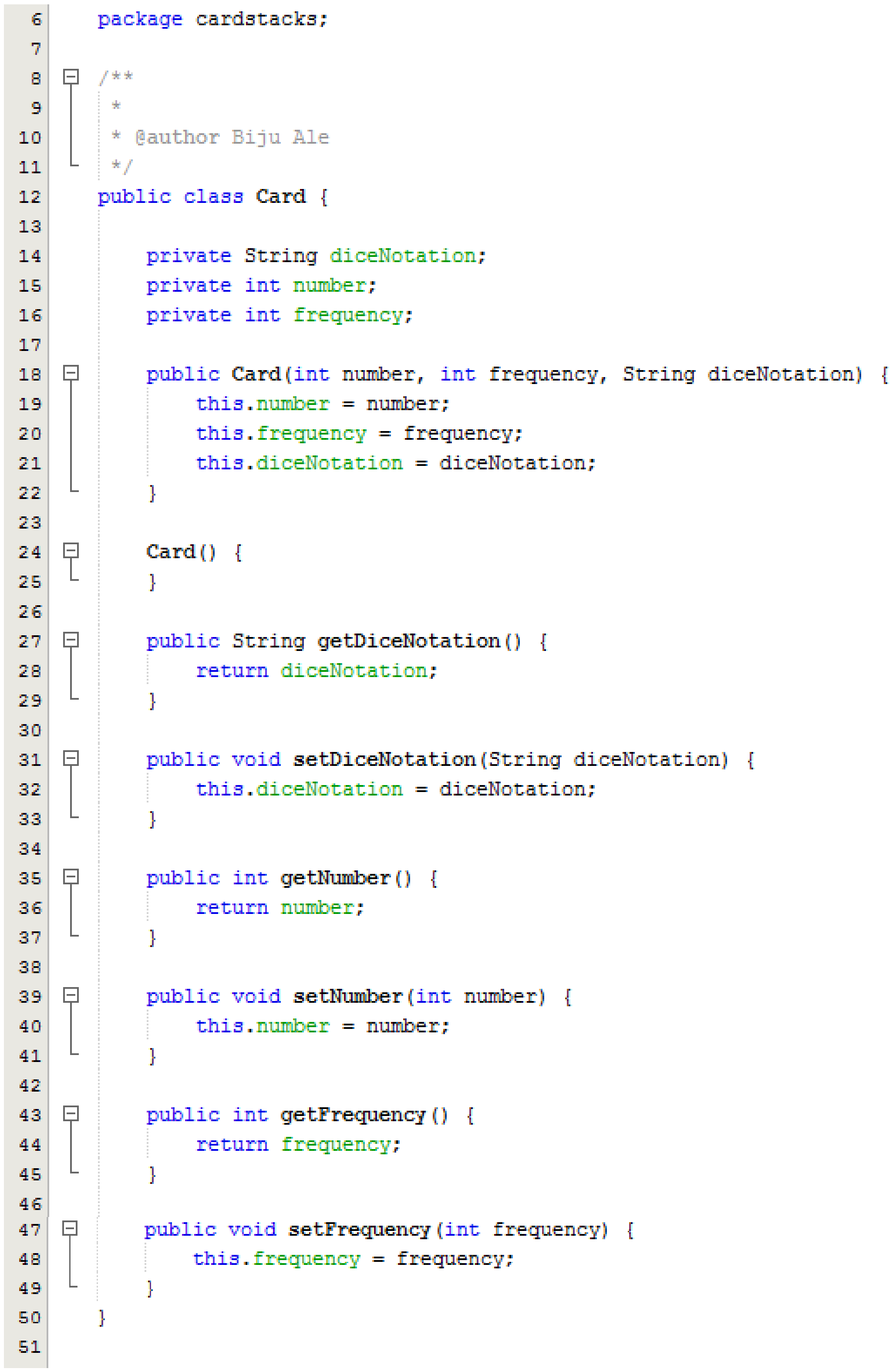
Class: HistoryDice



```
package cardstacks;

import java.util.ArrayList;

/**
 *
 * @author Biju Ale
 */
public class HistoryDice extends ArrayList<Dice> {

    public boolean addToDiceHistory(Dice dice) {
        for (Dice eachDice : this) {
            if (eachDice.getDiceName().equals(dice.getDiceName())) {
                return false;
            }
        }
        return add(dice);
    }
}
```

Class: Card

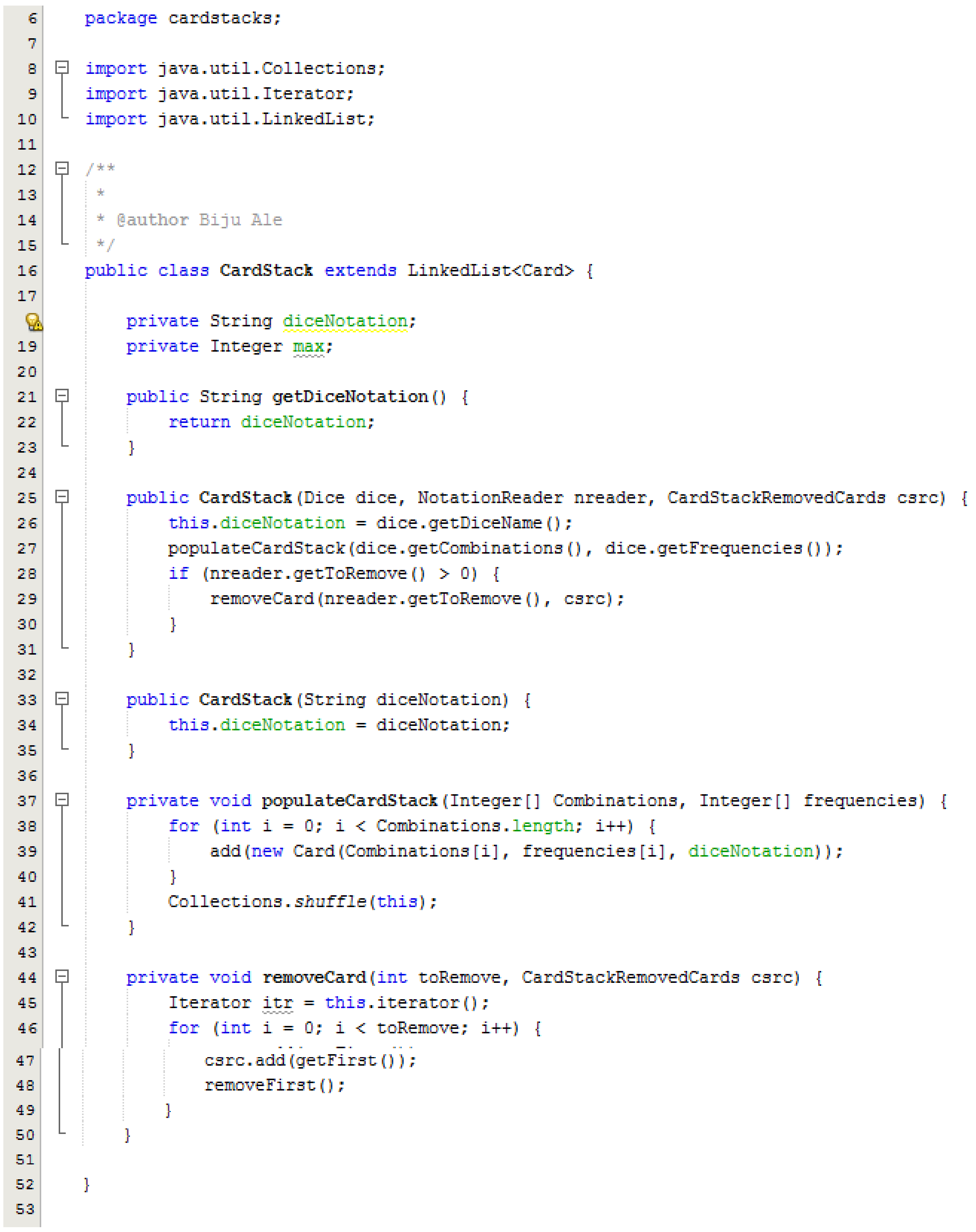
[PLEASE TURN OVER]



```java
package cardstacks;

/**
 *
 * @author Biju Ale
 */
public class Card {

    private String diceNotation;
    private int number;
    private int frequency;

    public Card(int number, int frequency, String diceNotation) {
        this.number = number;
        this.frequency = frequency;
        this.diceNotation = diceNotation;
    }

    Card() {
    }

    public String getDiceNotation() {
        return diceNotation;
    }

    public void setDiceNotation(String diceNotation) {
        this.diceNotation = diceNotation;
    }

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    public int getFrequency() {
        return frequency;
    }

    public void setFrequency(int frequency) {
        this.frequency = frequency;
    }
}

```

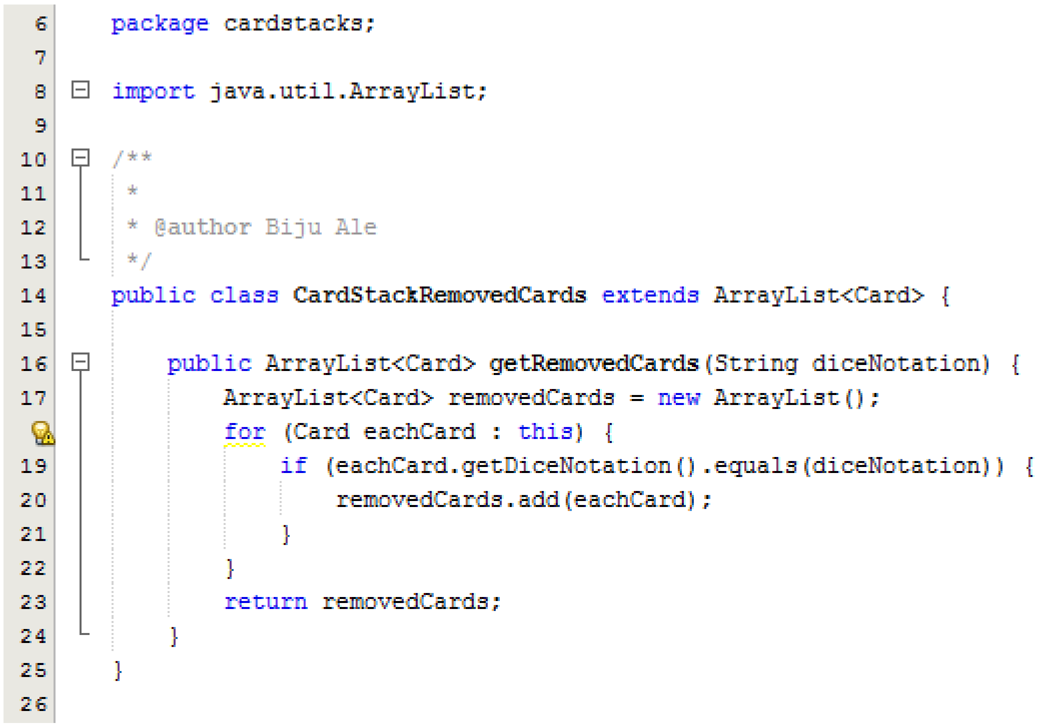
Class: CardStack

[PLEASE TURN OVER]



```java
package cardstacks;

import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedList;

/**
 *
 * @author Biju Ale
 */
public class CardStack extends LinkedList<Card> {

    private String diceNotation;
    private Integer max;

    public String getDiceNotation() {
        return diceNotation;
    }

    public CardStack(Dice dice, NotationReader nreader, CardStackRemovedCards csrc) {
        this.diceNotation = dice.getDiceName();
        populateCardStack(dice.getCombinations(), dice.getFrequencies());
        if (nreader.getToRemove() > 0) {
            removeCard(nreader.getToRemove(), csrc);
        }
    }

    public CardStack(String diceNotation) {
        this.diceNotation = diceNotation;
    }

    private void populateCardStack(Integer[] Combinations, Integer[] frequencies) {
        for (int i = 0; i < Combinations.length; i++) {
            add(new Card(Combinations[i], frequencies[i], diceNotation));
        }
        Collections.shuffle(this);
    }

    private void removeCard(int toRemove, CardStackRemovedCards csrc) {
        Iterator itr = this.iterator();
        for (int i = 0; i < toRemove; i++) {
            csrc.add(getFirst());
            removeFirst();
        }
    }

}
```

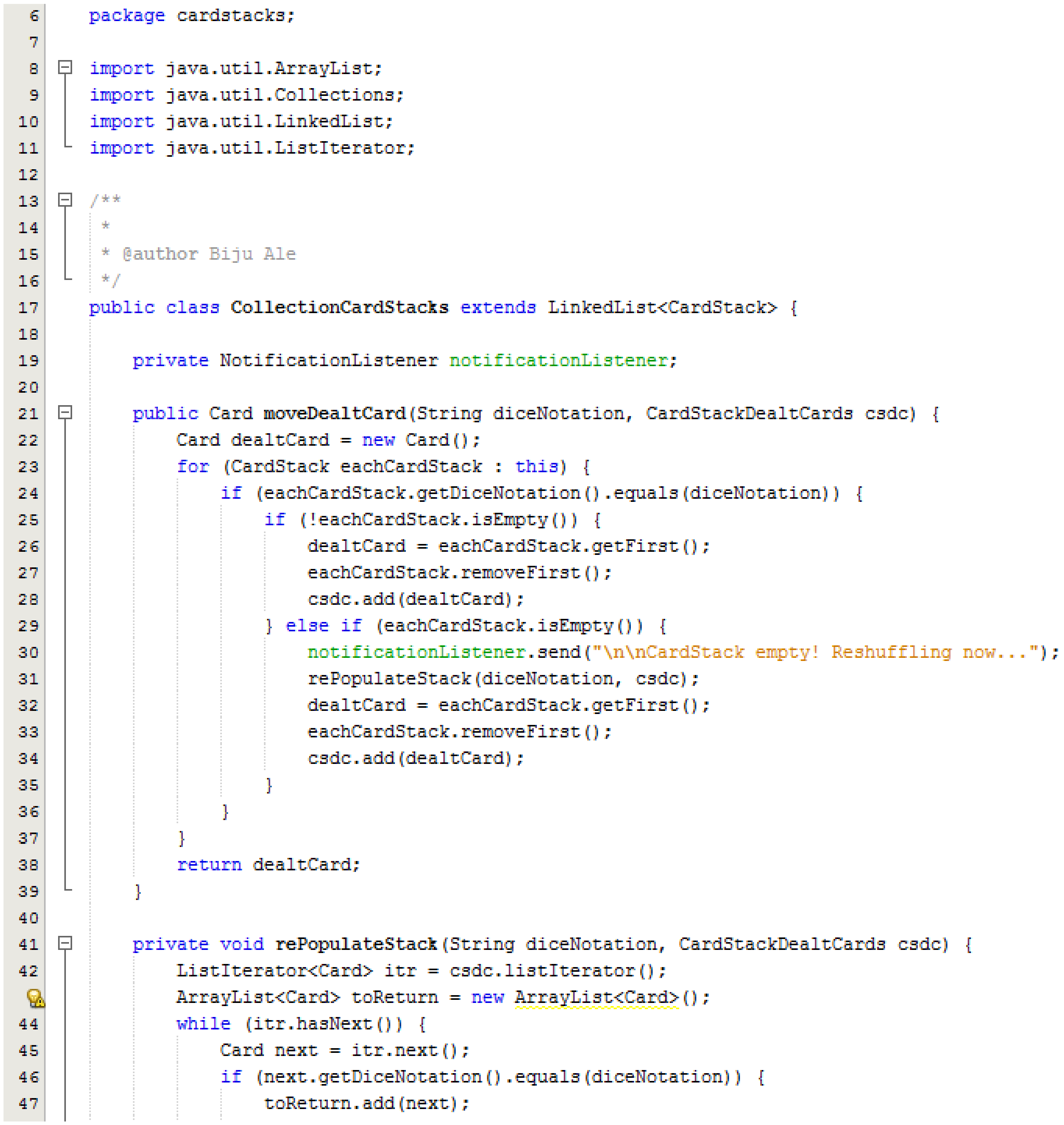
Class: CardStackRemovedCards



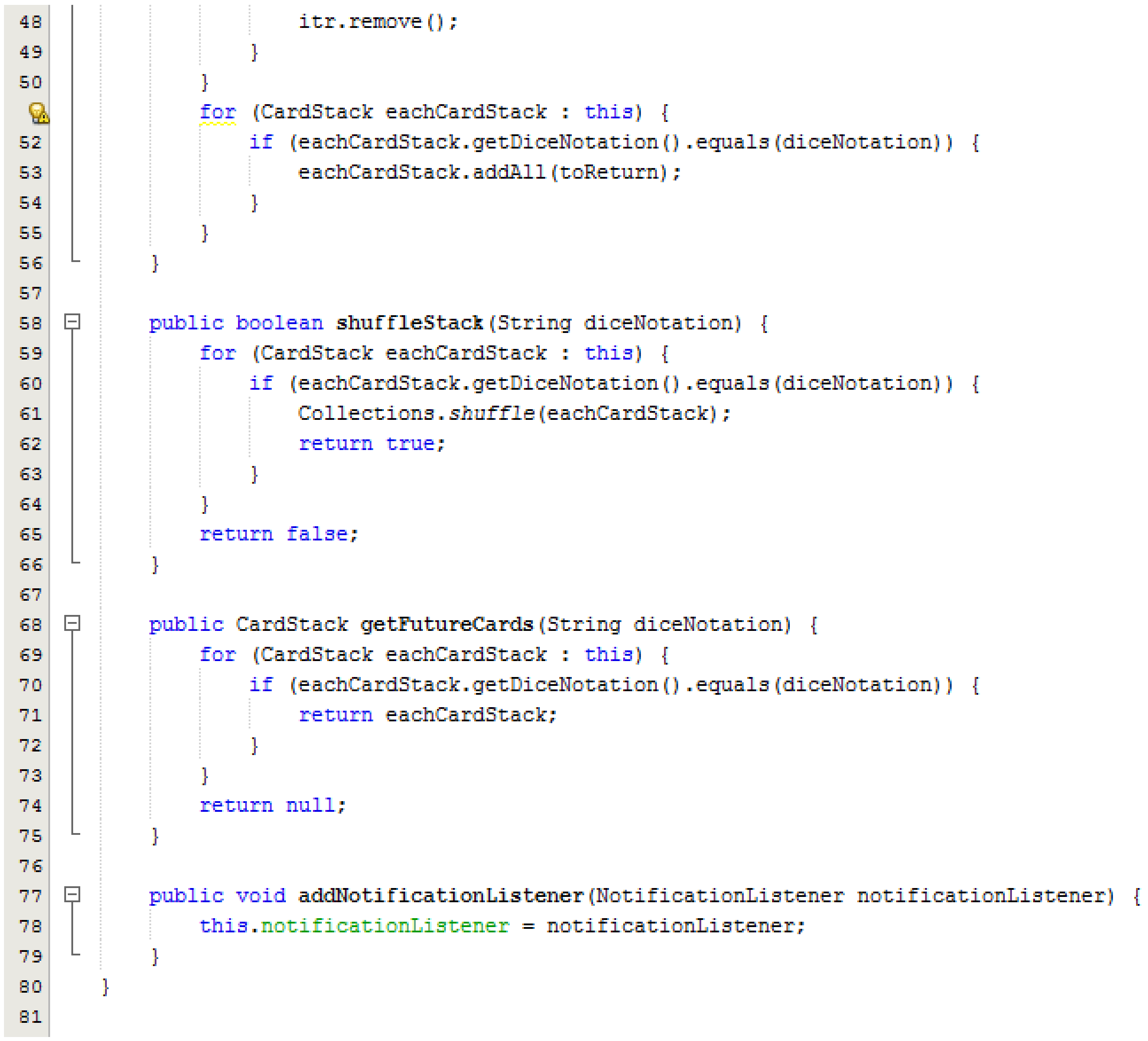
```
package cardstacks;

import java.util.ArrayList;

/**
 *
 * @author Biju Ale
 */
public class CardStackRemovedCards extends ArrayList<Card> {

    public ArrayList<Card> getRemovedCards(String diceNotation) {
        ArrayList<Card> removedCards = new ArrayList();
        for (Card eachCard : this) {
            if (eachCard.getDiceNotation().equals(diceNotation)) {
                removedCards.add(eachCard);
            }
        }
        return removedCards;
    }
}
```

Class: CollectionCardStacks

[PLEASE TURN OVER]

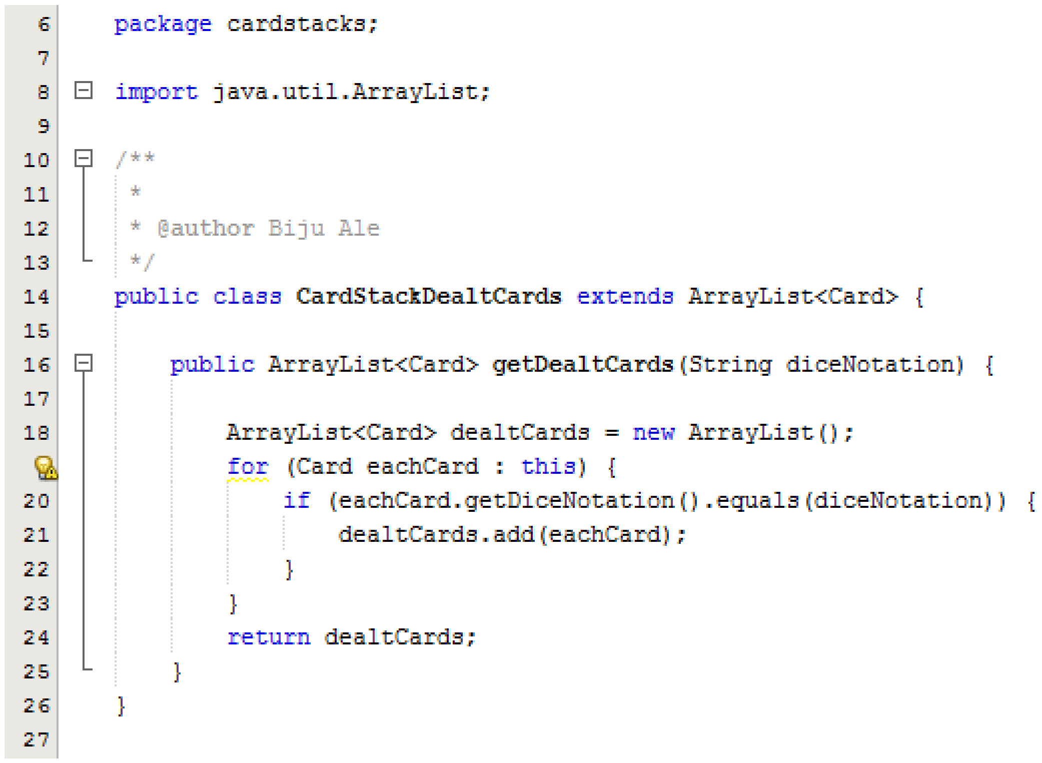


```java
package cardstacks;

import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedList;
import java.util.ListIterator;

/**
 *
 * @author Biju Ale
 */
public class CollectionCardStacks extends LinkedList<CardStack> {

    private NotificationListener notificationListener;

    public Card moveDealtCard(String diceNotation, CardStackDealtCards csdc) {
        Card dealtCard = new Card();
        for (CardStack eachCardStack : this) {
            if (eachCardStack.getDiceNotation().equals(diceNotation)) {
                if (!eachCardStack.isEmpty()) {
                    dealtCard = eachCardStack.getFirst();
                    eachCardStack.removeFirst();
                    csdc.add(dealtCard);
                } else if (eachCardStack.isEmpty()) {
                    notificationListener.send("\n\nCardStack empty! Reshuffling now...");
                    rePopulateStack(diceNotation, csdc);
                    dealtCard = eachCardStack.getFirst();
                    eachCardStack.removeFirst();
                    csdc.add(dealtCard);
                }
            }
        }
        return dealtCard;
    }

    private void rePopulateStack(String diceNotation, CardStackDealtCards csdc) {
        ListIterator<Card> itr = csdc.listIterator();
        ArrayList<Card> toReturn = new ArrayList<Card>();
        while (itr.hasNext()) {
            Card next = itr.next();
            if (next.getDiceNotation().equals(diceNotation)) {
                toReturn.add(next);
```



```
                itr.remove();
            }
        }
        for (CardStack eachCardStack : this) {
            if (eachCardStack.getDiceNotation().equals(diceNotation)) {
                eachCardStack.addAll(toReturn);
            }
        }
    }

    public boolean shuffleStack(String diceNotation) {
        for (CardStack eachCardStack : this) {
            if (eachCardStack.getDiceNotation().equals(diceNotation)) {
                Collections.shuffle(eachCardStack);
                return true;
            }
        }
        return false;
    }

    public CardStack getFutureCards(String diceNotation) {
        for (CardStack eachCardStack : this) {
            if (eachCardStack.getDiceNotation().equals(diceNotation)) {
                return eachCardStack;
            }
        }
        return null;
    }

    public void addNotificationListener(NotificationListener notificationListener) {
        this.notificationListener = notificationListener;
    }
}
```

Class: CardStackDealtCards

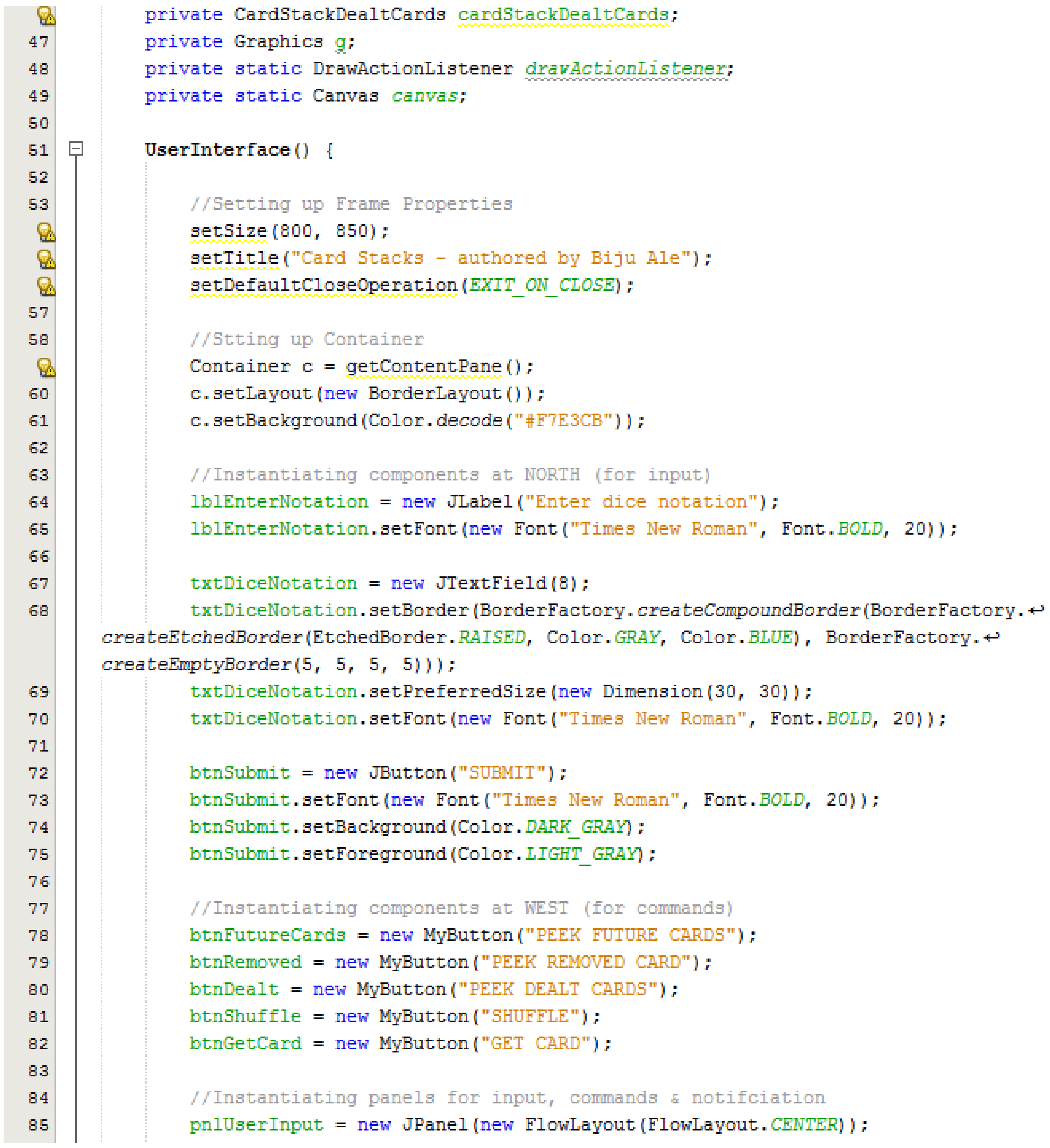


```java
package cardstacks;

import java.util.ArrayList;

/**
 *
 * @author Biju Ale
 */
public class CardStackDealtCards extends ArrayList<Card> {

    public ArrayList<Card> getDealtCards(String diceNotation) {

        ArrayList<Card> dealtCards = new ArrayList();
        for (Card eachCard : this) {
            if (eachCard.getDiceNotation().equals(diceNotation)) {
                dealtCards.add(eachCard);
            }
        }
        return dealtCards;
    }
}
```

Class: UserInterface

[PLEASE TURN OVER]

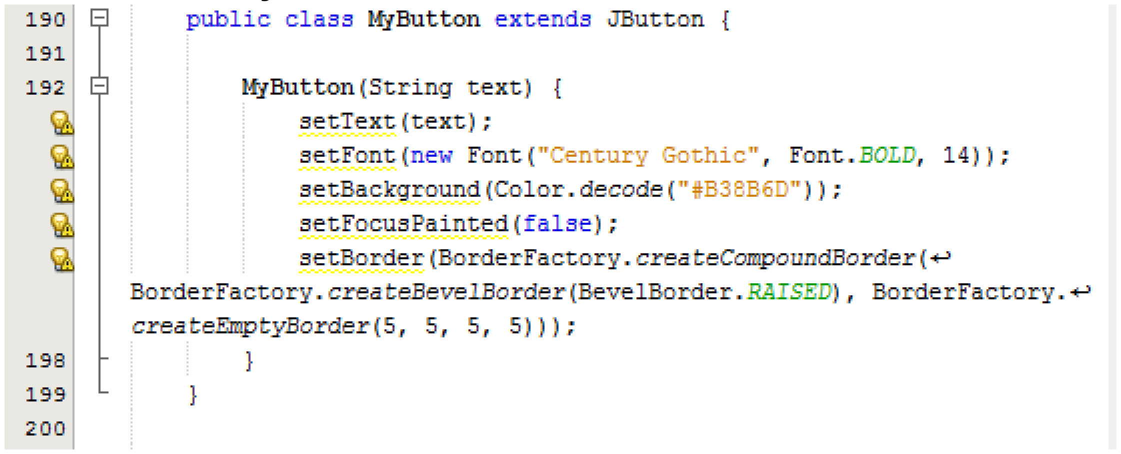
```java
package cardstacks;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.border.BevelBorder;
import javax.swing.border.EtchedBorder;

/**
 *
 * @author Biju Ale
 */
public class UserInterface extends JFrame implements ActionListener, NotificationListener {

    private JLabel lblEnterNotation;
    private JTextField txtDiceNotation;
    private static JTextArea txtNotification;
    private JButton btnSubmit, btnFutureCards, btnRemoved, btnDealt, btnShuffle, btnGetCard;
    private JPanel pnlUserInput, pnlCommands, pnlNotification;

    private NotationReader nreader;
    private HistoryDice historyDice;
    private CardStack cardStack;
    private CollectionCardStacks collectionCardStacks;
    private CardStackRemovedCards cardStackRemovedCards;
```



```java
    private CardStackDealtCards cardStackDealtCards;
    private Graphics g;
    private static DrawActionListener drawActionListener;
    private static Canvas canvas;

    UserInterface() {

        //Setting up Frame Properties
        setSize(800, 850);
        setTitle("Card Stacks - authored by Biju Ale");
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        //Stting up Container
        Container c = getContentPane();
        c.setLayout(new BorderLayout());
        c.setBackground(Color.decode("#F7E3CB"));

        //Instantiating components at NORTH (for input)
        lblEnterNotation = new JLabel("Enter dice notation");
        lblEnterNotation.setFont(new Font("Times New Roman", Font.BOLD, 20));

        txtDiceNotation = new JTextField(8);
        txtDiceNotation.setBorder(BorderFactory.createCompoundBorder(BorderFactory.↩
createEtchedBorder(EtchedBorder.RAISED, Color.GRAY, Color.BLUE), BorderFactory.↩
createEmptyBorder(5, 5, 5, 5)));
        txtDiceNotation.setPreferredSize(new Dimension(30, 30));
        txtDiceNotation.setFont(new Font("Times New Roman", Font.BOLD, 20));

        btnSubmit = new JButton("SUBMIT");
        btnSubmit.setFont(new Font("Times New Roman", Font.BOLD, 20));
        btnSubmit.setBackground(Color.DARK_GRAY);
        btnSubmit.setForeground(Color.LIGHT_GRAY);

        //Instantiating components at WEST (for commands)
        btnFutureCards = new MyButton("PEEK FUTURE CARDS");
        btnRemoved = new MyButton("PEEK REMOVED CARD");
        btnDealt = new MyButton("PEEK DEALT CARDS");
        btnShuffle = new MyButton("SHUFFLE");
        btnGetCard = new MyButton("GET CARD");

        //Instantiating panels for input, commands & notifciation
        pnlUserInput = new JPanel(new FlowLayout(FlowLayout.CENTER));
        pnlUserInput.setBackground(Color.decode("#1DE9B6"));
        pnlCommands = new JPanel(new GridLayout(5, 1));
        pnlNotification = new JPanel(new GridLayout(1, 1));

        //Adding all components to input panel
        pnlUserInput.add(lblEnterNotation);
        pnlUserInput.add(txtDiceNotation);
        pnlUserInput.add(btnSubmit);

        //Adding all components to commands panel
        pnlCommands.add(btnGetCard);
        pnlCommands.add(btnShuffle);
        pnlCommands.add(btnDealt);
        pnlCommands.add(btnRemoved);
        pnlCommands.add(btnFutureCards);

        //Adding notification area
        txtNotification = new JTextArea(3, 50);
        txtNotification.setFont(new Font("Times New Roman", Font.PLAIN, 20));
        txtNotification.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));
        txtNotification.setBackground(Color.decode("#1DE9B6"));
        JScrollPane txtNotificationScroll = new JScrollPane(txtNotification);

        c.add(pnlUserInput, BorderLayout.PAGE_START);
        c.add(pnlCommands, BorderLayout.LINE_START);
        c.add(txtNotificationScroll, BorderLayout.PAGE_END);

        //Adding action listener to all buttons
        btnSubmit.addActionListener(this);
        btnShuffle.addActionListener(this);
        btnRemoved.addActionListener(this);
        btnFutureCards.addActionListener(this);
        btnGetCard.addActionListener(this);
        btnDealt.addActionListener(this);

        //Initialize all stacks, dice & cardstack history
        historyDice = new HistoryDice();
        cardStackRemovedCards = new CardStackRemovedCards();
        cardStackDealtCards = new CardStackDealtCards();
        collectionCardStacks = new CollectionCardStacks();
        collectionCardStacks.addNotificationListener(this);

        //Instantiating canvas
        canvas = new Canvas();
        canvas.addNotificationListener(this);
        canvas.setPreferredSize(new Dimension(6000, 2000));
        canvas.setBackground(Color.decode("#F7E3CB"));
        JScrollPane canvasScroll = new JScrollPane(canvas);
        c.add(canvasScroll, BorderLayout.CENTER);

        drawActionListener = canvas;
        setLocationRelativeTo(null);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        JButton btnSrc = (JButton) e.getSource();
        if (btnSrc.equals(btnSubmit)) {
            String diceNotation = txtDiceNotation.getText();
            try {
                nreader = new NotationReader();
                nreader.parseDiceNotation(diceNotation);
                Dice dice = new Dice(nreader);
                if (historyDice.addToDiceHistory(dice)) {
                    cardStack = new CardStack(dice, nreader, cardStackRemovedCards);
                    collectionCardStacks.add(cardStack);
                    txtNotification.append("\n\nRolling...\n" + diceNotation + "'s ↩
combinations & frequencies added to card stack.");
                    btnGetCard.doClick();
                } else {
                    txtNotification.append("\n\nAlready rolled dice - " + diceNotation);
                    btnGetCard.doClick();
                }
            } catch (Exception ex) {
                txtNotification.append(ex.getMessage());
                txtDiceNotation.requestFocus();
            }
        } else if (btnSrc.equals(btnGetCard)) {
            canvas.sendSingleDealtCard(collectionCardStacks.moveDealtCard(↩
txtDiceNotation.getText(), cardStackDealtCards));
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_GET_CARD;
        } else if (btnSrc.equals(btnDealt)) {
            canvas.sendAllDealtCards(cardStackDealtCards.getDealtCards(txtDiceNotation.↩
```

```java
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_DEALT_CARD;
        } else if (btnSrc.equals(btnRemoved)) {
            canvas.sendRemovedCards(cardStackRemovedCards.getRemovedCards(txtDiceNotation.getText()));
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_REMOVED;
        } else if (btnSrc.equals(btnFutureCards)) {
            canvas.sendFutureCards(collectionCardStacks.getFutureCards(txtDiceNotation.getText()));
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_FUTURE;
        } else if (btnSrc.equals(btnShuffle)) {
            if (collectionCardStacks.shuffleStack(txtDiceNotation.getText())) {
                txtNotification.append("\n\nShuffing stack " + txtDiceNotation.getText() + " complete.");
                canvas.repaint();
            } else {
                txtNotification.append("\n\nNo such stack to shuffle.");
            }
        }
    }

    @Override
    public void send(String notification
    ) {
        txtNotification.append(notification);
    }

    public class MyButton extends JButton {

        MyButton(String text) {
            setText(text);
            setFont(new Font("Century Gothic", Font.BOLD, 14));
            setBackground(Color.decode("#B38B6D"));
            setFocusPainted(false);
            setBorder(BorderFactory.createCompoundBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED), BorderFactory.createEmptyBorder(5, 5, 5, 5)));
        }
    }

    public static void main(String[] args) {
        UserInterface UI = new UserInterface();
        UI.setVisible(true);
    }
}
```

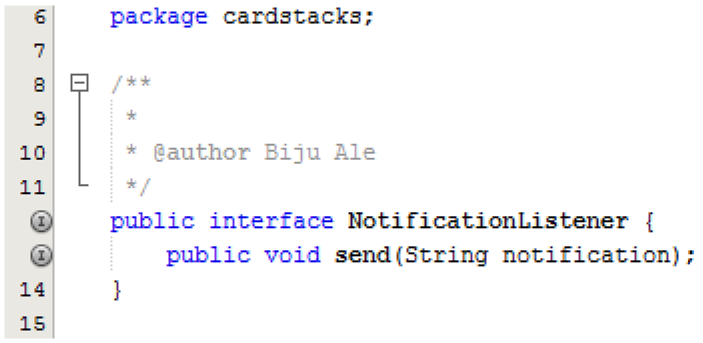
Inner Class: MyButton



```java
    public class MyButton extends JButton {

        MyButton(String text) {
            setText(text);
            setFont(new Font("Century Gothic", Font.BOLD, 14));
            setBackground(Color.decode("#B38B6D"));
            setFocusPainted(false);
            setBorder(BorderFactory.createCompoundBorder(↩
BorderFactory.createBevelBorder(BevelBorder.RAISED), BorderFactory.↩
createEmptyBorder(5, 5, 5, 5)));
        }
    }
```

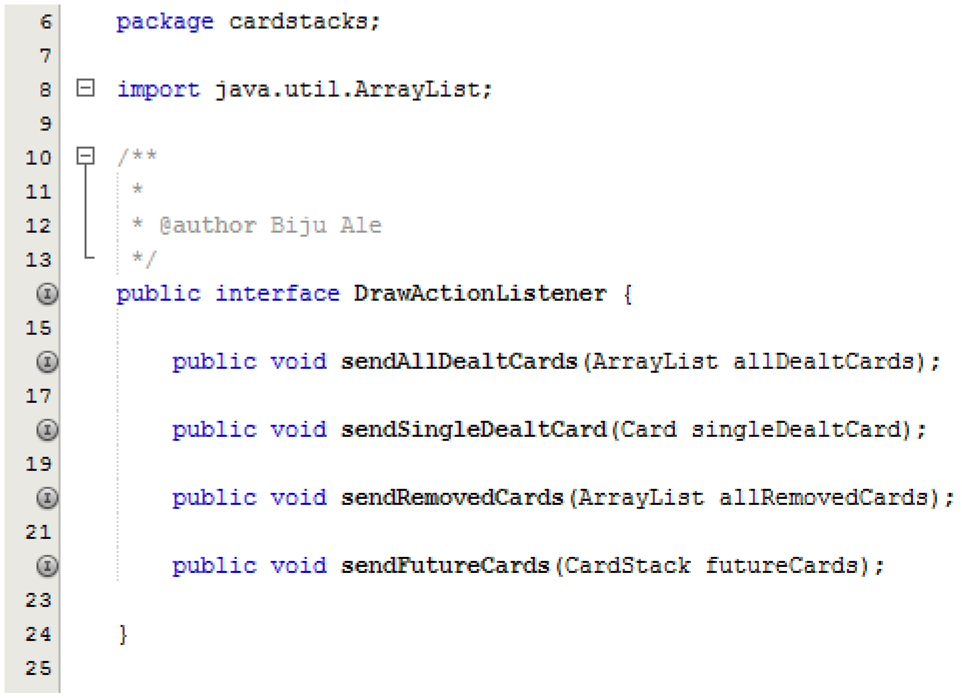
Nested inside Class: UserInterface.

Interface: NotificationListener



```java
package cardstacks;

/**
 *
 * @author Biju Ale
 */
public interface NotificationListener {
    public void send(String notification);
}
```

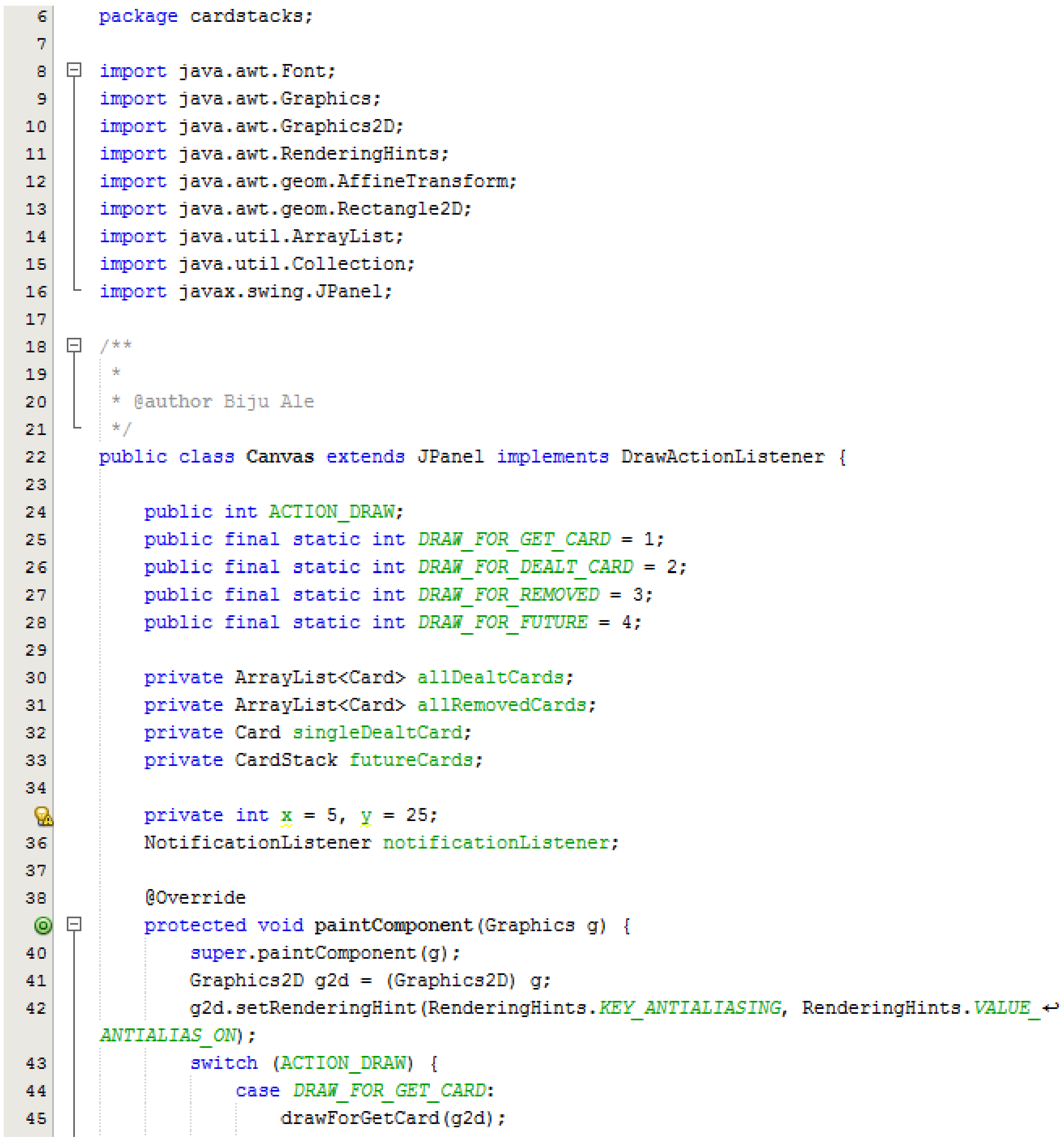
Interface: DrawActionListener



```java
package cardstacks;

import java.util.ArrayList;

/**
 *
 * @author Biju Ale
 */
public interface DrawActionListener {

    public void sendAllDealtCards(ArrayList allDealtCards);

    public void sendSingleDealtCard(Card singleDealtCard);

    public void sendRemovedCards(ArrayList allRemovedCards);

    public void sendFutureCards(CardStack futureCards);

}
```

Class: Canvas

[PLEASE TURN OVER]



```java
package cardstacks;

import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.geom.AffineTransform;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import java.util.Collection;
import javax.swing.JPanel;

/**
 *
 * @author Biju Ale
 */
public class Canvas extends JPanel implements DrawActionListener {

    public int ACTION_DRAW;
    public final static int DRAW_FOR_GET_CARD = 1;
    public final static int DRAW_FOR_DEALT_CARD = 2;
    public final static int DRAW_FOR_REMOVED = 3;
    public final static int DRAW_FOR_FUTURE = 4;

    private ArrayList<Card> allDealtCards;
    private ArrayList<Card> allRemovedCards;
    private Card singleDealtCard;
    private CardStack futureCards;

    private int x = 5, y = 25;
    NotificationListener notificationListener;

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        switch (ACTION_DRAW) {
            case DRAW_FOR_GET_CARD:
                drawForGetCard(g2d);
                break;
            case DRAW_FOR_REMOVED:
                drawForPeekRemovedCard(g2d);
                break;
            case DRAW_FOR_DEALT_CARD:
                drawForPeekDealtCard(g2d);
                break;
            case DRAW_FOR_FUTURE:
                drawForPeekFutureCard(g2d);
                break;
        }
    }

    private void drawForGetCard(Graphics g2d) {
        if (singleDealtCard.getDiceNotation() != null) {
            notificationListener.send("\n\nGetting card...");
            g2d.setFont(new Font("Times New Roman", Font.PLAIN, 25));
            g2d.drawString("Recently dealt card for stack: " + singleDealtCard.↩
getDiceNotation(), x, y);

            Font font = new Font("Times New Roman", Font.PLAIN, 200);
            g2d.setFont(font);
            String num = "" + singleDealtCard.getNumber();
            g2d.drawString(num, 250, 292);

            Font font1 = new Font("Times New Roman", Font.PLAIN, 50);
            g2d.setFont(font1);
            String freq = "" + (double) Math.round((singleDealtCard.getFrequency() / ↩
100d) * 10d) / 10d + "%";
            g2d.drawString(freq, 250, 394);
        } else {
            notificationListener.send("\n\nNo card exists to deal! Enter valid notation"↩
);
        }
    }

    private void drawForPeekDealtCard(Graphics g2d) {
        if (allDealtCards != null && allDealtCards.size() > 0) {
            notificationListener.send("\n\nGetting dealt cards with Graph...");
            int x = 5;
            g2d.setFont(new Font("Times New Roman", Font.PLAIN, 20));
            g2d.drawString("All Dealt Cards from stack: " + allDealtCards.get(0).↩
getDiceNotation(), x, y);

            for (Card eachDealtCard : allDealtCards) {
                g2d.drawString("" + eachDealtCard.getNumber() + " [" + (double) Math.↩
round((eachDealtCard.getFrequency() / 100d) * 10d) / 10d + "%], ", x, 50);
                x += 90;
            }
            drawGraph(g2d, "Dealt Cards", allDealtCards);
        } else {
            notificationListener.send("\n\nNo dealt card exists to peek!");
        }
    }

    private void drawForPeekRemovedCard(Graphics g2d) {
        if (allRemovedCards != null && allRemovedCards.size() > 0) {
            notificationListener.send("\n\nGetting removed cards...");
            int x = 5;
            int y = 250;
            g2d.setFont(new Font("Times New Roman", Font.PLAIN, 25));
            g2d.drawString("All Removed Cards from stack: " + allRemovedCards.get(0).↩
getDiceNotation(), x, y);

            for (Card eachRemovedCard : allRemovedCards) {
                g2d.drawString("" + eachRemovedCard.getNumber() + " [" + (double) Math.↩
round((eachRemovedCard.getFrequency() / 100d) * 10d) / 10d + "%], ", x, y + 40);
                x += 110;
            }
        } else {
            notificationListener.send("\n\nNo removed card exists to peek!");
        }
    }

    private void drawForPeekFutureCard(Graphics g2d) {
        if (futureCards != null && futureCards.size() > 0) {
            notificationListener.send("\n\nGetting furure cards with Graph...");
            int x = 5;
            g2d.setFont(new Font("Times New Roman", Font.PLAIN, 20));
            g2d.drawString("Future Cards from stack: " + futureCards.getFirst().↩
getDiceNotation(), x, y);
```

```java

                for (Card eachFutureCard : futureCards) {
                    g2d.drawString("" + eachFutureCard.getNumber() + " [" + (double) Math.round((eachFutureCard.getFrequency() / 100d) * 10d) / 10d + "%], ", x, 50);
                    x += 90;
                }
                drawGraph(g2d, "Future Cards", futureCards);
            } else {
                notificationListener.send("\n\nNo future cards exists left to peek!");
            }

    }

    private void drawGraph(Graphics g, String cardType, Collection<Card> stack) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        int x = 40, y = 100;

        //Y-Coordinate lines
        int s = y;
        for (int i = y; i > -10; i = i - 10) {
            g2d.drawString("" + (i), x - 10, s + 8);
            g2d.drawLine(x + 20, s, getWidth(), s);
            s += 40;
        }

        //X-Coordinate label
        g2d.setFont(new Font("Times New Roman", Font.PLAIN, 20));
//          g2d.setColor(Color.decode("#371C00"));
        g2d.drawString(cardType + " (Card number)", 350, 550);

        //Y-Coordinate label
        AffineTransform t = new AffineTransform();
        t.rotate(Math.toRadians(-90));
        g2d.setFont(new Font("Times New Roman", Font.PLAIN, 20).deriveFont(t));
//          g2d.setColor(Color.decode("#371C00"));
        g2d.drawString("Chance of occurence (%)", x - 17, 380);

        //Bars
        double chartHeight = 400;
        double barWidth = 15;
        for (Card eachCard : stack) {
            double barHeight = (double) eachCard.getFrequency() / (double) Dice.ROLL_TIMES * 100 * 4d;
            double newY = chartHeight - barHeight;

            Rectangle2D bar = new Rectangle2D.Double(x + 50, newY + y, barWidth, barHeight);
            g2d.fill(bar);
            AffineTransform t1 = new AffineTransform();
            t.rotate(Math.toRadians(-90));
            g2d.setFont(new Font("Times New Roman", Font.ITALIC, 12).deriveFont(t1));
            g2d.drawString("" + eachCard.getNumber(), (float) x + 50, (float) chartHeight + 120);
            x += 30;
        }
    }

    @Override
    public void sendAllDealtCards(ArrayList allDealtCards) {
        this.allDealtCards = allDealtCards;
        this.repaint();
    }

    @Override
    public void sendSingleDealtCard(Card singleDealtCard) {
        this.singleDealtCard = singleDealtCard;
        this.repaint();
    }

    @Override
    public void sendRemovedCards(ArrayList allRemovedCards) {
        this.allRemovedCards = allRemovedCards;
        this.repaint();
    }

    @Override
    public void sendFutureCards(CardStack futureCards) {
        this.futureCards = futureCards;
        this.repaint();
    }

    public void addNotificationListener(NotificationListener notificationListener) {
        this.notificationListener = notificationListener;
    }
}
```

Task-2 (Testing Data)

Test Suite No.1

Testingclass: cardstacks.NotationReader
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: • 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: • 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!"

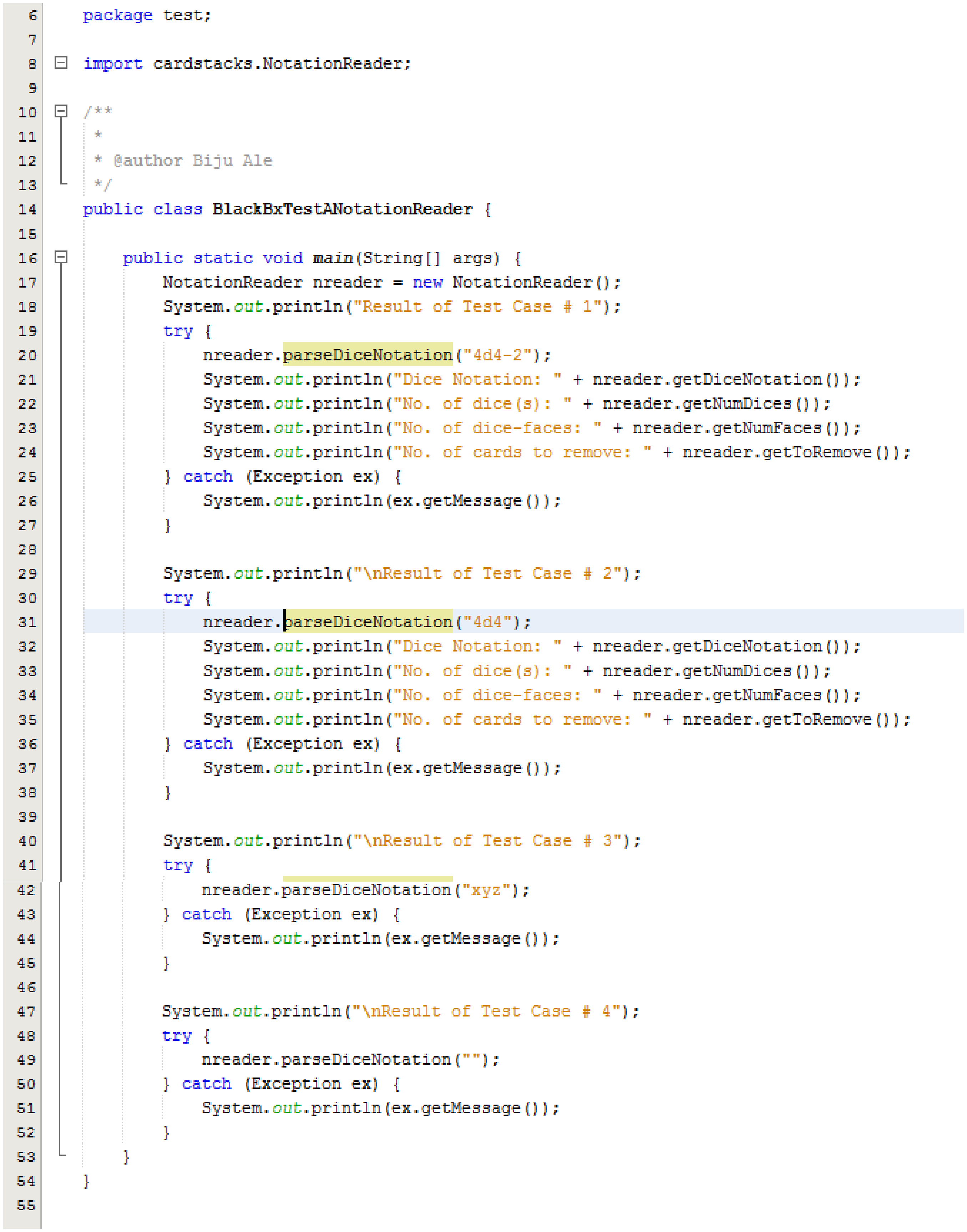
Method of Equivalence partitioning:

- 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

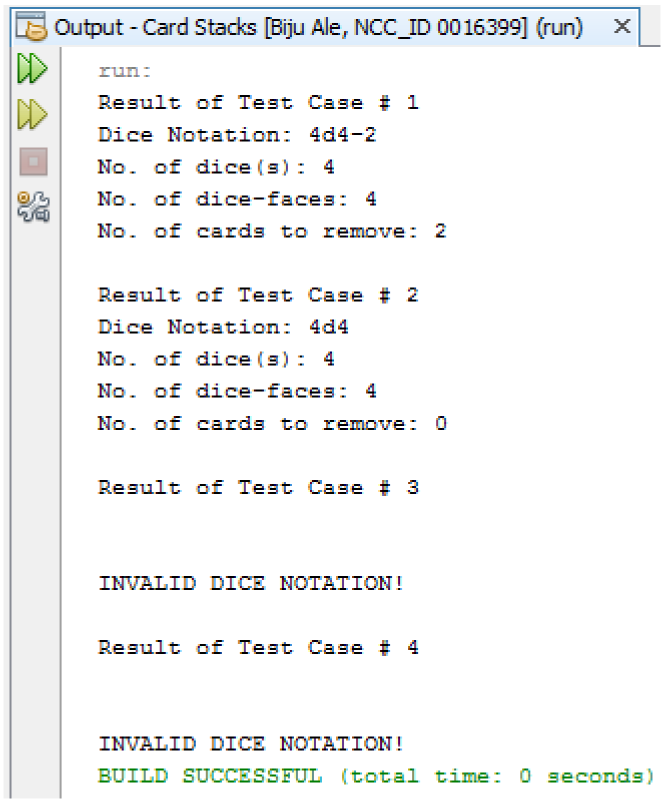
Source Code

[PLEASE TURN OVER]



```java
package test;

import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class BlackBxTestANotationReader {

    public static void main(String[] args) {
        NotationReader nreader = new NotationReader();
        System.out.println("Result of Test Case # 1");
        try {
            nreader.parseDiceNotation("4d4-2");
            System.out.println("Dice Notation: " + nreader.getDiceNotation());
            System.out.println("No. of dice(s): " + nreader.getNumDices());
            System.out.println("No. of dice-faces: " + nreader.getNumFaces());
            System.out.println("No. of cards to remove: " + nreader.getToRemove());
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

        System.out.println("\nResult of Test Case # 2");
        try {
            nreader.parseDiceNotation("4d4");
            System.out.println("Dice Notation: " + nreader.getDiceNotation());
            System.out.println("No. of dice(s): " + nreader.getNumDices());
            System.out.println("No. of dice-faces: " + nreader.getNumFaces());
            System.out.println("No. of cards to remove: " + nreader.getToRemove());
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

        System.out.println("\nResult of Test Case # 3");
        try {
            nreader.parseDiceNotation("xyz");
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

        System.out.println("\nResult of Test Case # 4");
        try {
            nreader.parseDiceNotation("");
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

Output


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

```
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: • 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: • 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.

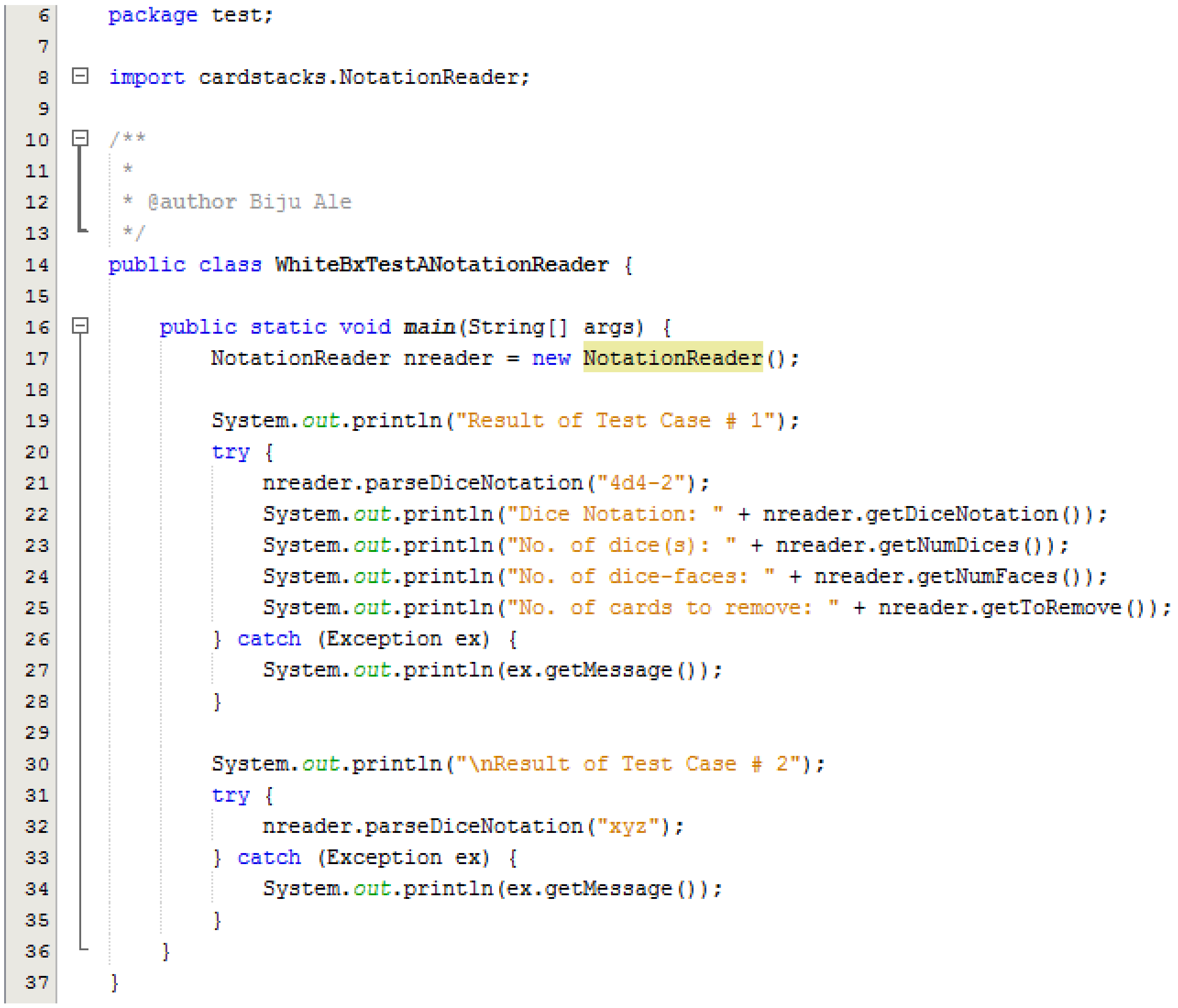
Test Suite No.2

Testingclass: cardstacks.NotationReader
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: • diceNotation = 4d4-2 • numDices = 4 • numFaces = 4 • toRemove = 2
2	void parseDiceNotation(String diceNotation)	xyz	Invalid	Throws NumberFormatException with message - "INVALID DICE NOTATION!"

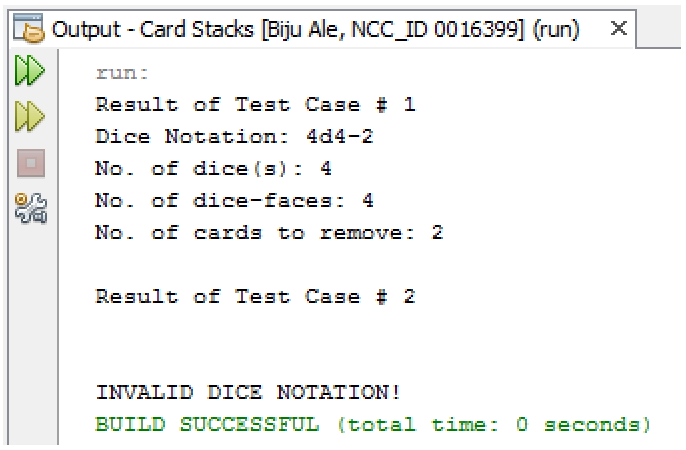
Test Execution

<u>Source Code</u>



```java
package test;

import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class WhiteBxTestANotationReader {

    public static void main(String[] args) {
        NotationReader nreader = new NotationReader();

        System.out.println("Result of Test Case # 1");
        try {
            nreader.parseDiceNotation("4d4-2");
            System.out.println("Dice Notation: " + nreader.getDiceNotation());
            System.out.println("No. of dice(s): " + nreader.getNumDices());
            System.out.println("No. of dice-faces: " + nreader.getNumFaces());
            System.out.println("No. of cards to remove: " + nreader.getToRemove());
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

        System.out.println("\nResult of Test Case # 2");
        try {
            nreader.parseDiceNotation("xyz");
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

<u>Output</u>



```
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(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' 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.

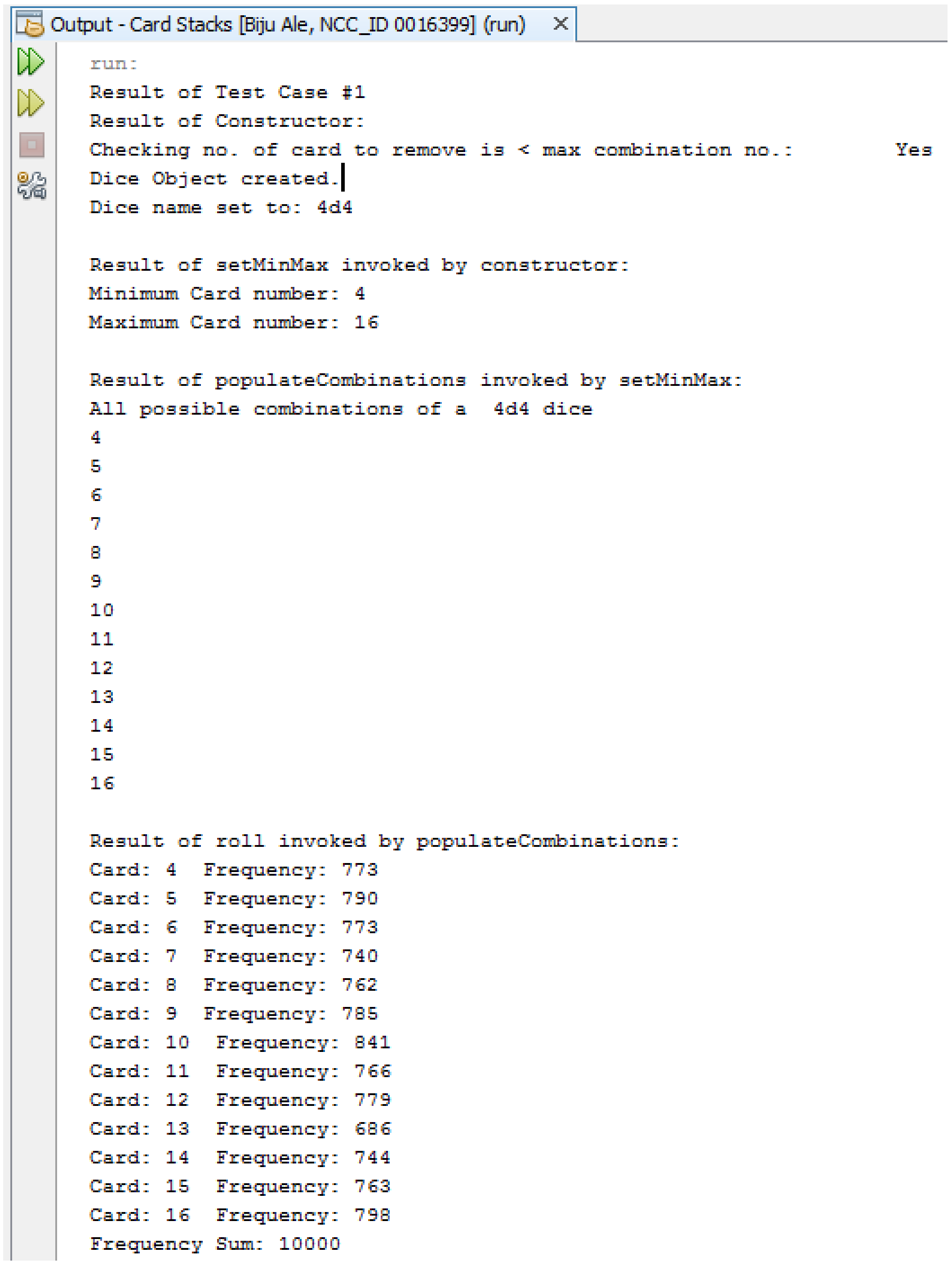
Test Execution

<u>Source Code</u>

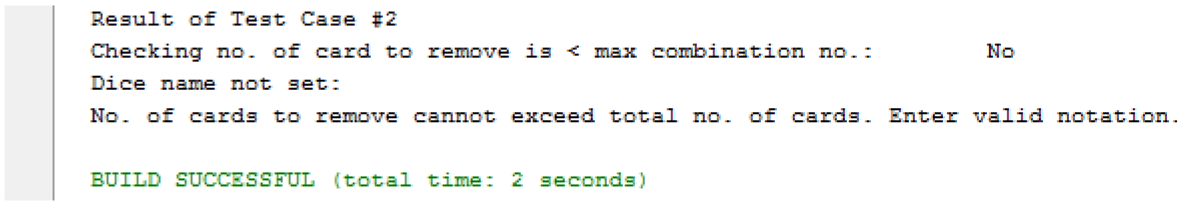
[PLEASE TURN OVER]

```java
package test;

import cardstacks.Dice;
import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class BlackBxTestADice {

    public static void main(String[] args) {
        System.out.println("Result of Test Case #1");
        NotationReader nreader = new NotationReader();
        try {
            nreader.parseDiceNotation("4d4");
            Dice dice = new Dice(nreader);
            //Testing Constructor
            System.out.println("Result of Constructor:");
            System.out.println("Checking no. of card to remove is < max combination no.:\tYes");
            System.out.println("Dice Object created.");
            System.out.println("Dice name set to: " + dice.getDiceName());

            //Testing setMinMax invoked by constructor
            System.out.println("\nResult of setMinMax invoked by constructor:");
            System.out.println("Minimum Card number: " + dice.getMinCombination());
            System.out.println("Maximum Card number: " + dice.getMaxCombination());

            //Testing populateCombinations invoked by setMinMax
            System.out.println("\nResult of populateCombinations invoked by setMinMax:");
            System.out.println("All possible combinations of a  4d4 dice");
            for (Integer eachCombination : dice.getCombinations()) {
                System.out.println(eachCombination);
            }

            //Testing roll invoked by populateCombinations
            System.out.println("\nResult of roll invoked by populateCombinations:");
            int sumFreqquencies = 0;
            for (int i = 0; i < dice.getFrequencies().length; i++) {
                System.out.println("Card: " + dice.getCombinations()[i] + "  Frequency: " + dice.getFrequencies()[i]);
                sumFreqquencies += dice.getFrequencies()[i];
            }
            System.out.println("Frequency Sum: " + sumFreqquencies);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        System.out.println("\nResult of Test Case #2");
        try {
            nreader.parseDiceNotation("4d4-344");
            Dice dice = new Dice(nreader);
        } catch (Exception ex) {
            System.out.println("Checking no. of card to remove is < max combination no.:\tNo");
            System.out.println("Dice name not set: ");
            System.out.println(ex.getMessage());
        }
    }

}
```

Output


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

```
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(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.	Yes
2	Dice (NotificationReader nreader)	nreader	*Valid object (with 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"	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"

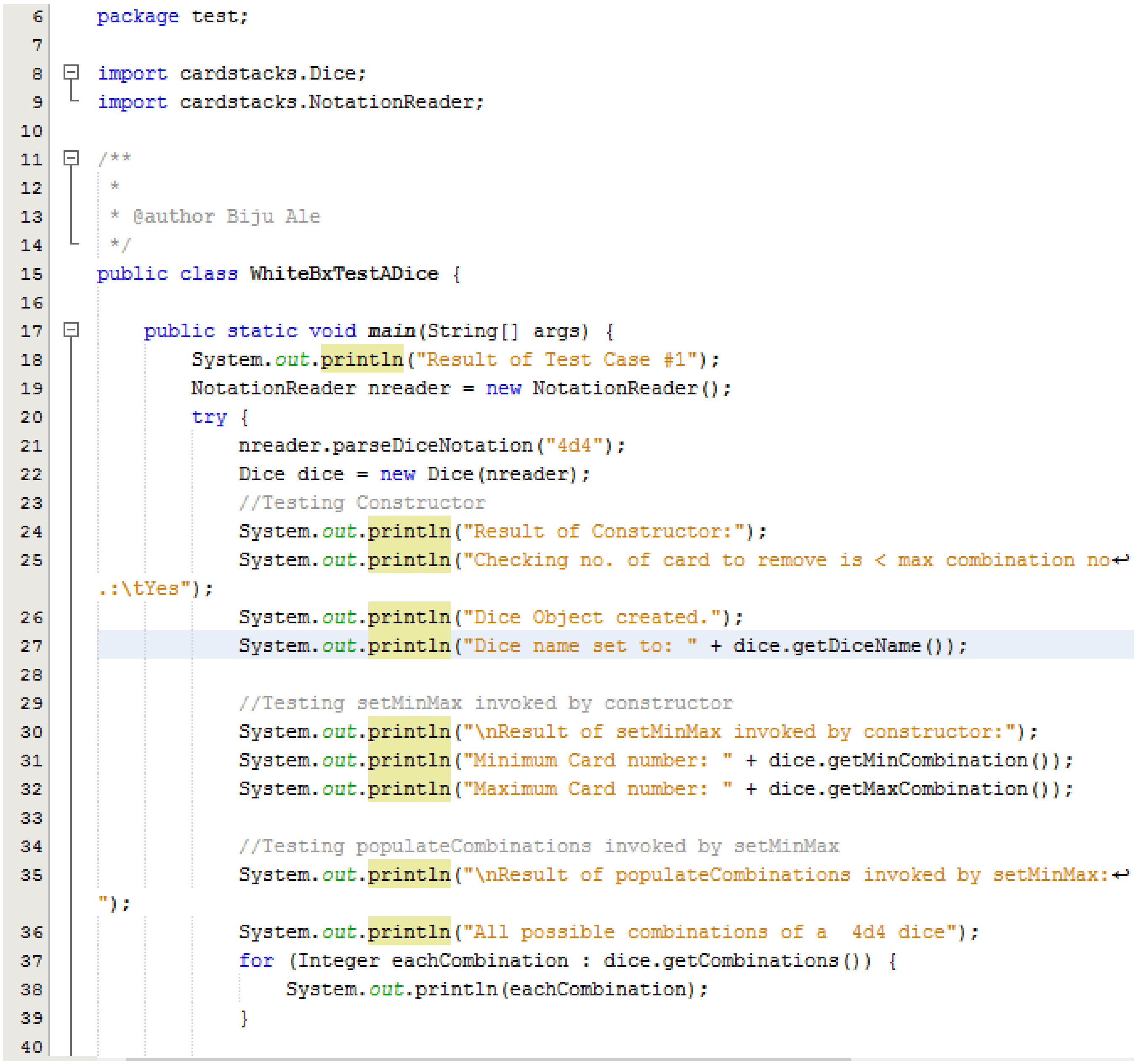
Method of test data selection:

- *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.

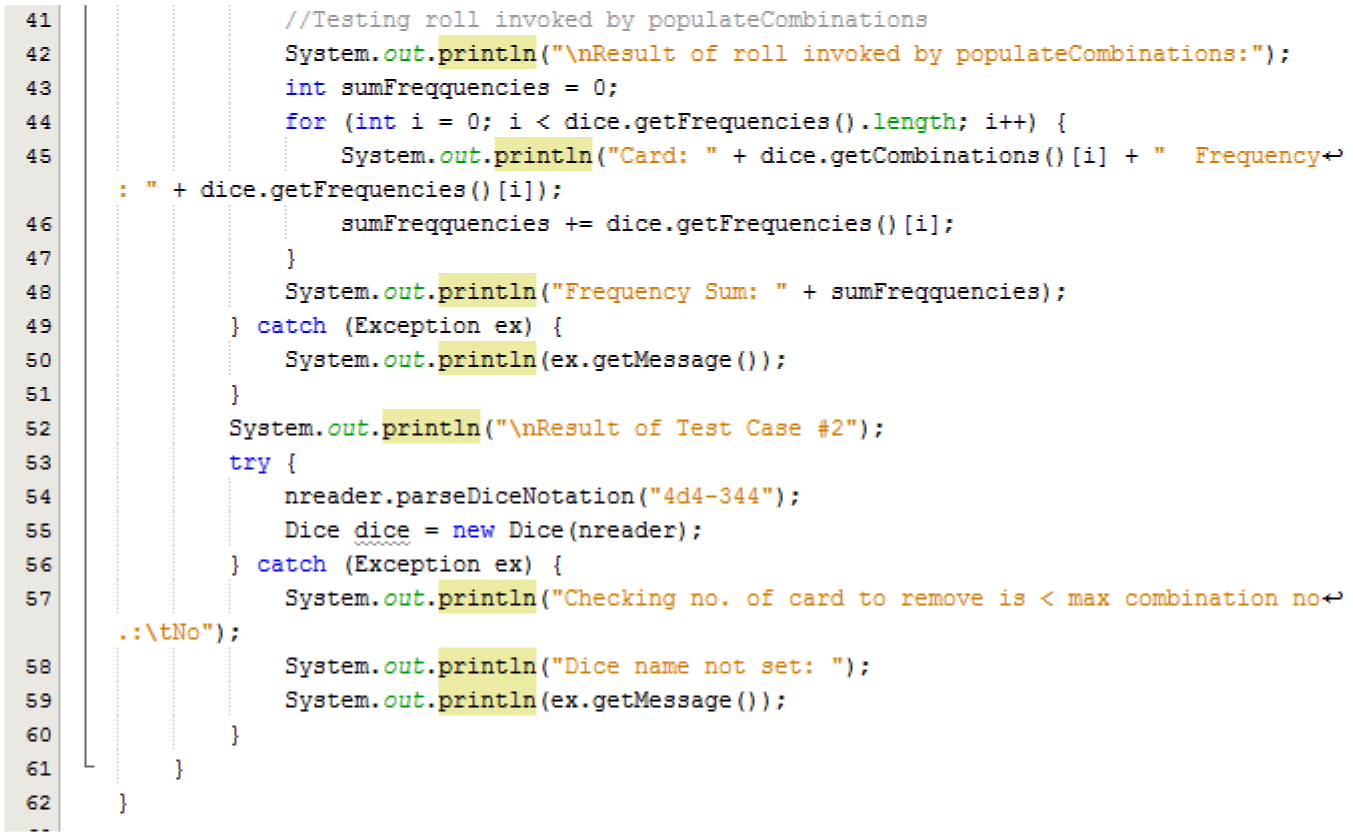
Test Execution

Source Code

[PLEASE TURN OVER]



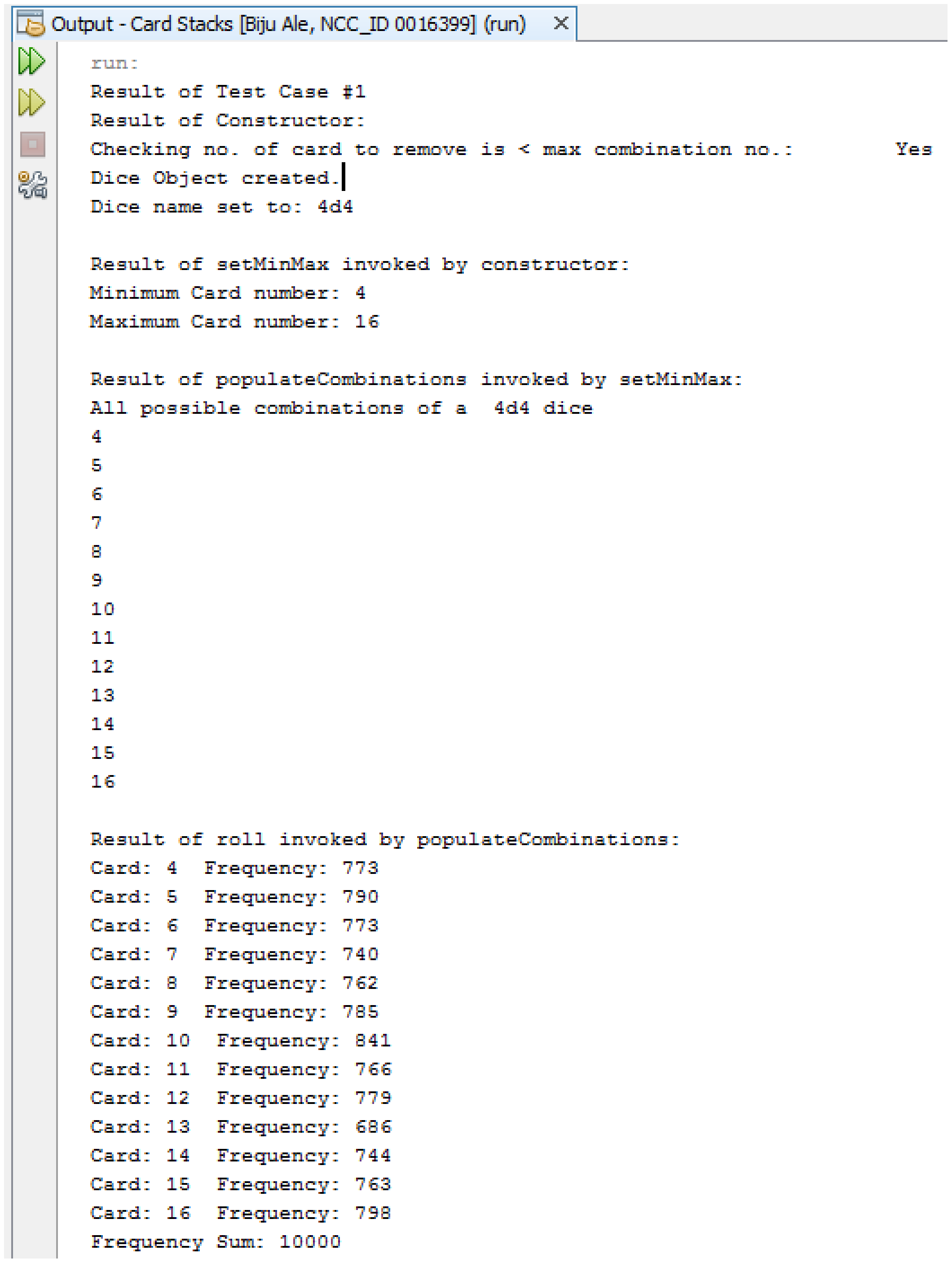
```java
package test;

import cardstacks.Dice;
import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class WhiteBxTestADice {

    public static void main(String[] args) {
        System.out.println("Result of Test Case #1");
        NotationReader nreader = new NotationReader();
        try {
            nreader.parseDiceNotation("4d4");
            Dice dice = new Dice(nreader);
            //Testing Constructor
            System.out.println("Result of Constructor:");
            System.out.println("Checking no. of card to remove is < max combination no.:\tYes");
            System.out.println("Dice Object created.");
            System.out.println("Dice name set to: " + dice.getDiceName());

            //Testing setMinMax invoked by constructor
            System.out.println("\nResult of setMinMax invoked by constructor:");
            System.out.println("Minimum Card number: " + dice.getMinCombination());
            System.out.println("Maximum Card number: " + dice.getMaxCombination());

            //Testing populateCombinations invoked by setMinMax
            System.out.println("\nResult of populateCombinations invoked by setMinMax:");
            System.out.println("All possible combinations of a  4d4 dice");
            for (Integer eachCombination : dice.getCombinations()) {
                System.out.println(eachCombination);
            }

```



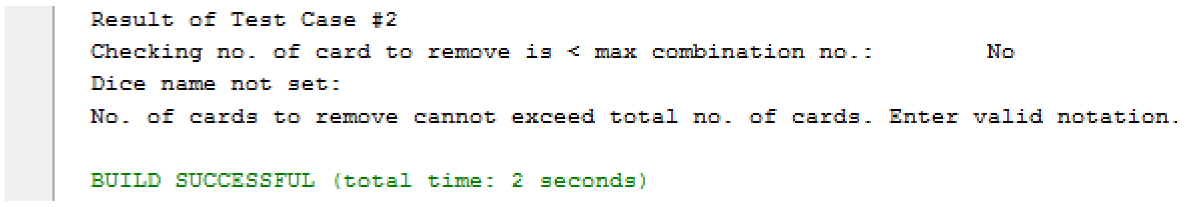
```
            //Testing roll invoked by populateCombinations
            System.out.println("\nResult of roll invoked by populateCombinations:");
            int sumFreqquencies = 0;
            for (int i = 0; i < dice.getFrequencies().length; i++) {
                System.out.println("Card: " + dice.getCombinations()[i] + "  Frequency: " + dice.getFrequencies()[i]);
                sumFreqquencies += dice.getFrequencies()[i];
            }
            System.out.println("Frequency Sum: " + sumFreqquencies);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        System.out.println("\nResult of Test Case #2");
        try {
            nreader.parseDiceNotation("4d4-344");
            Dice dice = new Dice(nreader);
        } catch (Exception ex) {
            System.out.println("Checking no. of card to remove is < max combination no.:\tNo");
            System.out.println("Dice name not set: ");
            System.out.println(ex.getMessage());
        }
    }
}
```

Output:

[PLEASE TURN OVER]


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

```
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

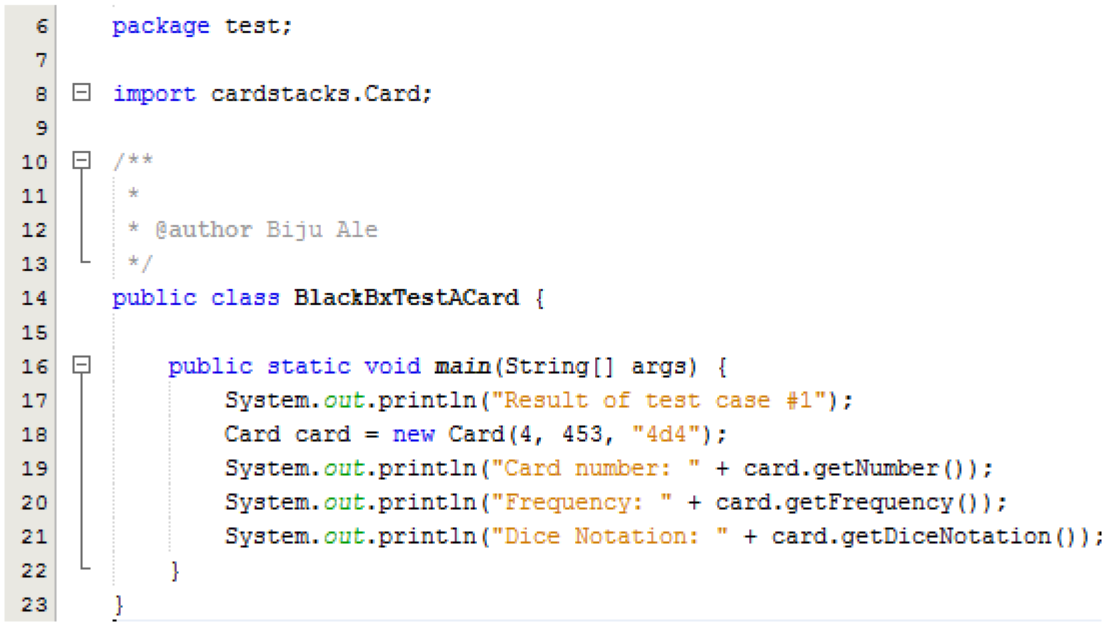
#	Method	Test Data	Input type	Expected Outcome
1	Card(int number, int frequency, String diceNotation)	(4,453,"4d4")	Valid	Constructor should set the values of instance variables as following: • number = 4 • frequency = 453 • diceNotation = 4d4

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.

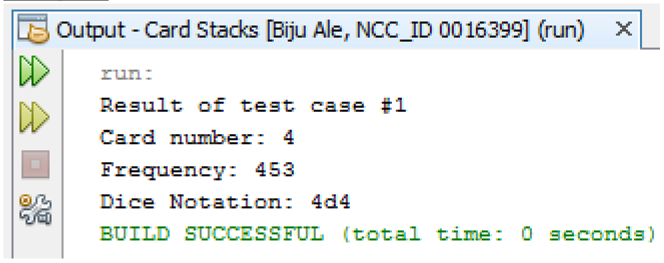
Test Execution

Source Code



```
package test;

import cardstacks.Card;

/**
 *
 * @author Biju Ale
 */
public class BlackBxTestACard {

    public static void main(String[] args) {
        System.out.println("Result of test case #1");
        Card card = new Card(4, 453, "4d4");
        System.out.println("Card number: " + card.getNumber());
        System.out.println("Frequency: " + card.getFrequency());
        System.out.println("Dice Notation: " + card.getDiceNotation());
    }
}
```

Output


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

```
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: • number = 4 • frequency = 453 • diceNotation = 4d4	Yes

Test Summary

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

Test Suite No.6

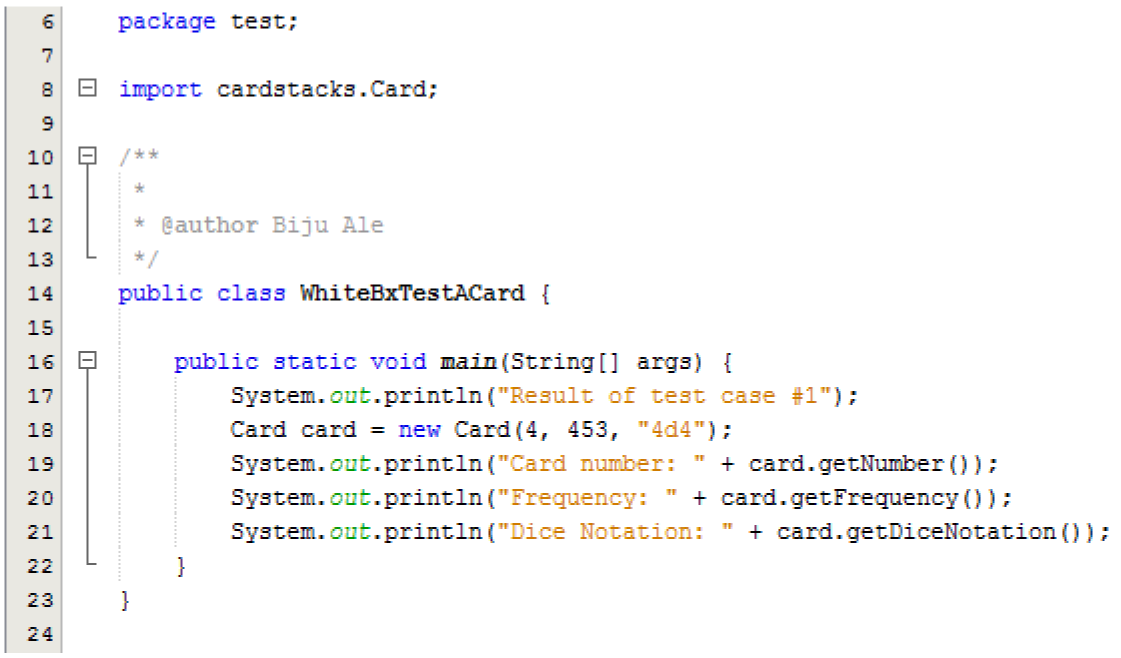
Testingclass: cardstacks.Dice
Testingtype: White Box / Unit Testing

#	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: • number = 4 • frequency = 453 • diceNotation = 4d4

Method of test data selection:

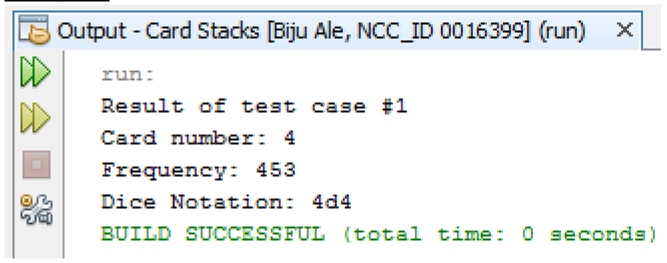
This constructor always receives valid input only, because it is only invoked after NotationReader calss that was checked in Test Suite No.1 & 2, validates the input from GUI.

Test Execution



```
package test;

import cardstacks.Card;

/**
 *
 * @author Biju Ale
 */
public class WhiteBxTestACard {

    public static void main(String[] args) {
        System.out.println("Result of test case #1");
        Card card = new Card(4, 453, "4d4");
        System.out.println("Card number: " + card.getNumber());
        System.out.println("Frequency: " + card.getFrequency());
        System.out.println("Dice Notation: " + card.getDiceNotation());
    }
}
```

Output



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

```
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.

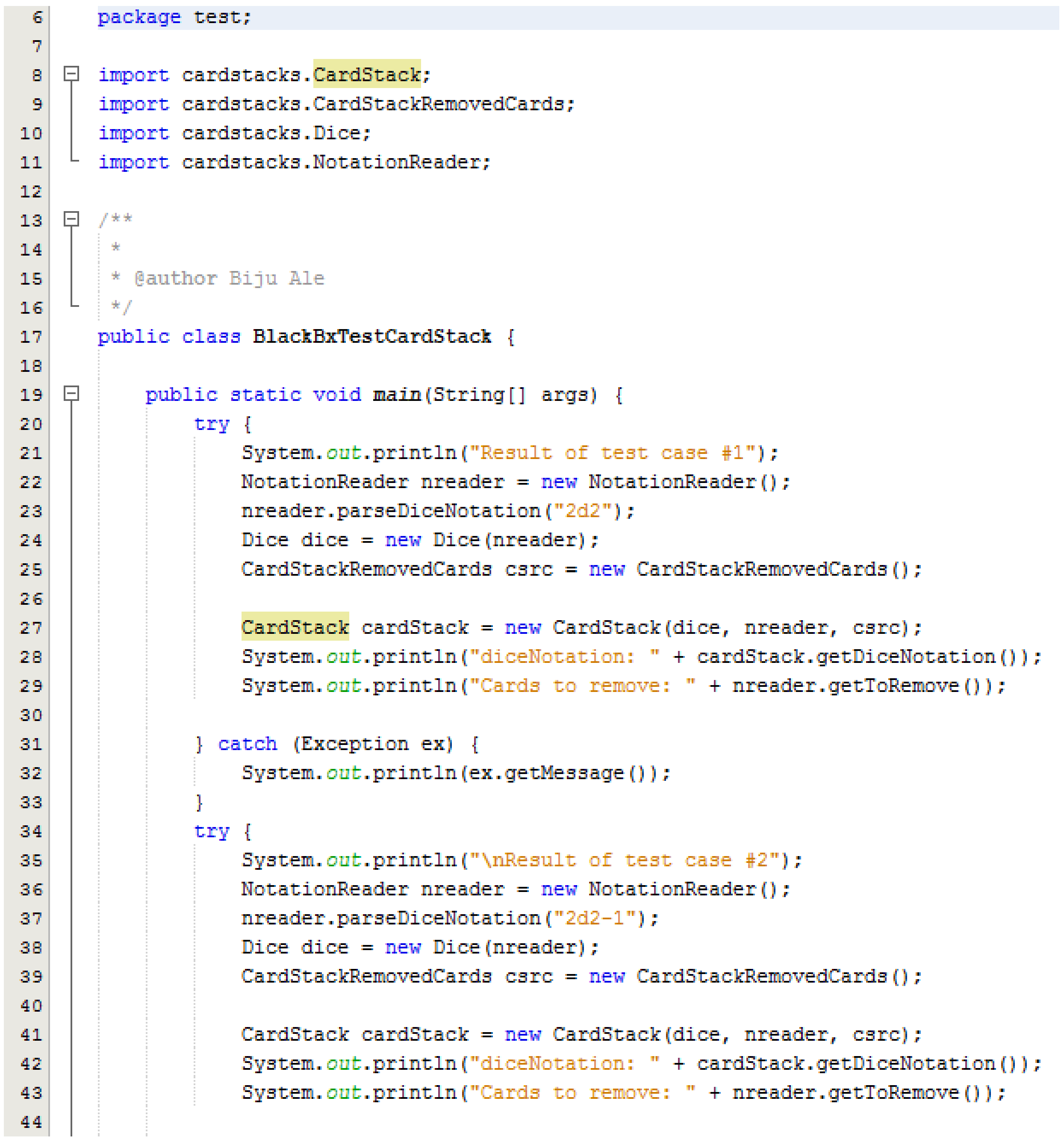
Method of test data selection:

- *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.

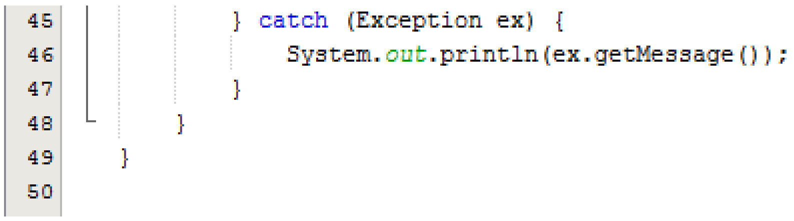
Test Execution

<u>Source Code</u>

[PLEASE TURN OVER]

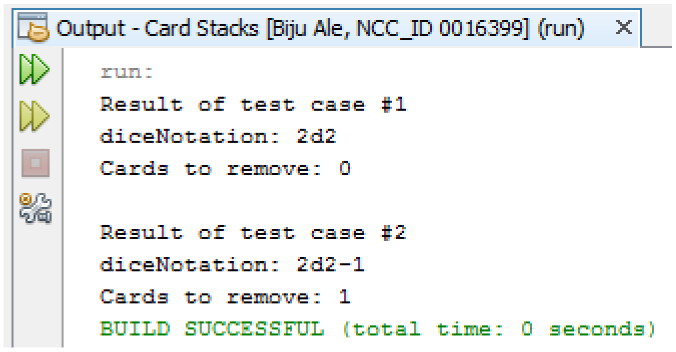


```java
package test;

import cardstacks.CardStack;
import cardstacks.CardStackRemovedCards;
import cardstacks.Dice;
import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class BlackBxTestCardStack {

    public static void main(String[] args) {
        try {
            System.out.println("Result of test case #1");
            NotationReader nreader = new NotationReader();
            nreader.parseDiceNotation("2d2");
            Dice dice = new Dice(nreader);
            CardStackRemovedCards csrc = new CardStackRemovedCards();

            CardStack cardStack = new CardStack(dice, nreader, csrc);
            System.out.println("diceNotation: " + cardStack.getDiceNotation());
            System.out.println("Cards to remove: " + nreader.getToRemove());

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        try {
            System.out.println("\nResult of test case #2");
            NotationReader nreader = new NotationReader();
            nreader.parseDiceNotation("2d2-1");
            Dice dice = new Dice(nreader);
            CardStackRemovedCards csrc = new CardStackRemovedCards();

            CardStack cardStack = new CardStack(dice, nreader, csrc);
            System.out.println("diceNotation: " + cardStack.getDiceNotation());
            System.out.println("Cards to remove: " + nreader.getToRemove());

```



```
            } catch (Exception ex) {
                System.out.println(ex.getMessage());
            }
        }
    }

```

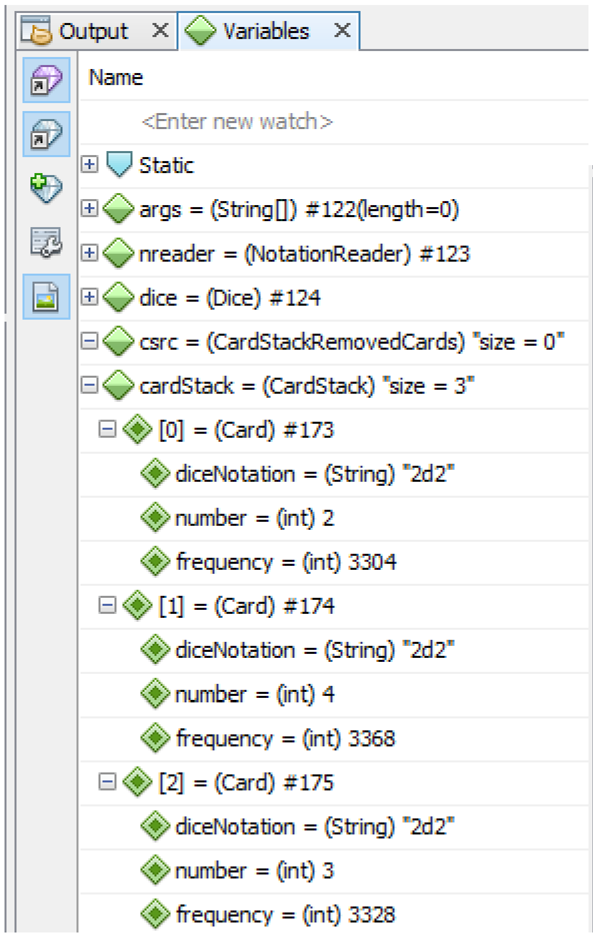
Output



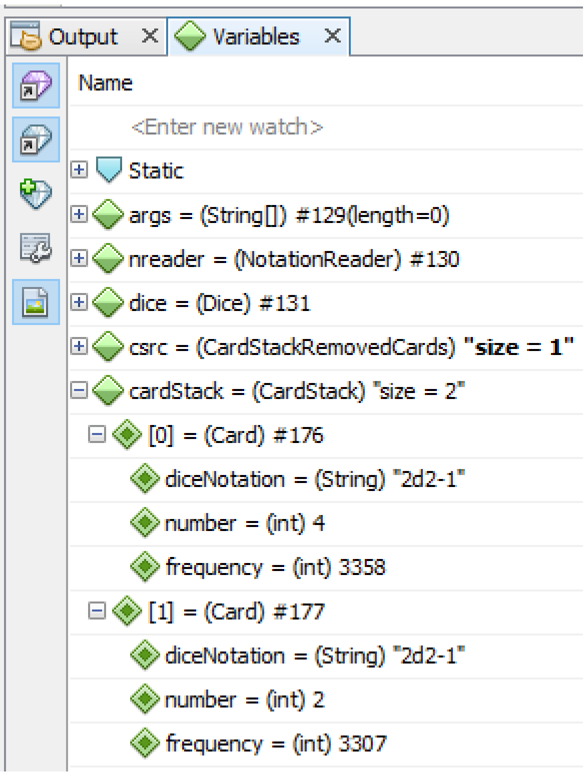
```
Output - Card Stacks [Biju Ale, NCC_ID 0016399] (run)
run:
Result of test case #1
diceNotation: 2d2
Cards to remove: 0

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:



```
Output  ✕  Variables  ✕
Name
<Enter new watch>
Static
args = (String[]) #122(length=0)
nreader = (NotationReader) #123
dice = (Dice) #124
csrc = (CardStackRemovedCards) "size = 0"
cardStack = (CardStack) "size = 3"
  [0] = (Card) #173
    diceNotation = (String) "2d2"
    number = (int) 2
    frequency = (int) 3304
  [1] = (Card) #174
    diceNotation = (String) "2d2"
    number = (int) 4
    frequency = (int) 3368
  [2] = (Card) #175
    diceNotation = (String) "2d2"
    number = (int) 3
    frequency = (int) 3328
```



```
Output  ✕  Variables  ✕
Name
<Enter new watch>
Static
args = (String[]) #129(length=0)
nreader = (NotationReader) #130
dice = (Dice) #131
csrc = (CardStackRemovedCards) "size = 1"
cardStack = (CardStack) "size = 2"
  [0] = (Card) #176
    diceNotation = (String) "2d2-1"
    number = (int) 4
    frequency = (int) 3358
  [1] = (Card) #177
    diceNotation = (String) "2d2-1"
    number = (int) 2
    frequency = (int) 3307
```

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.

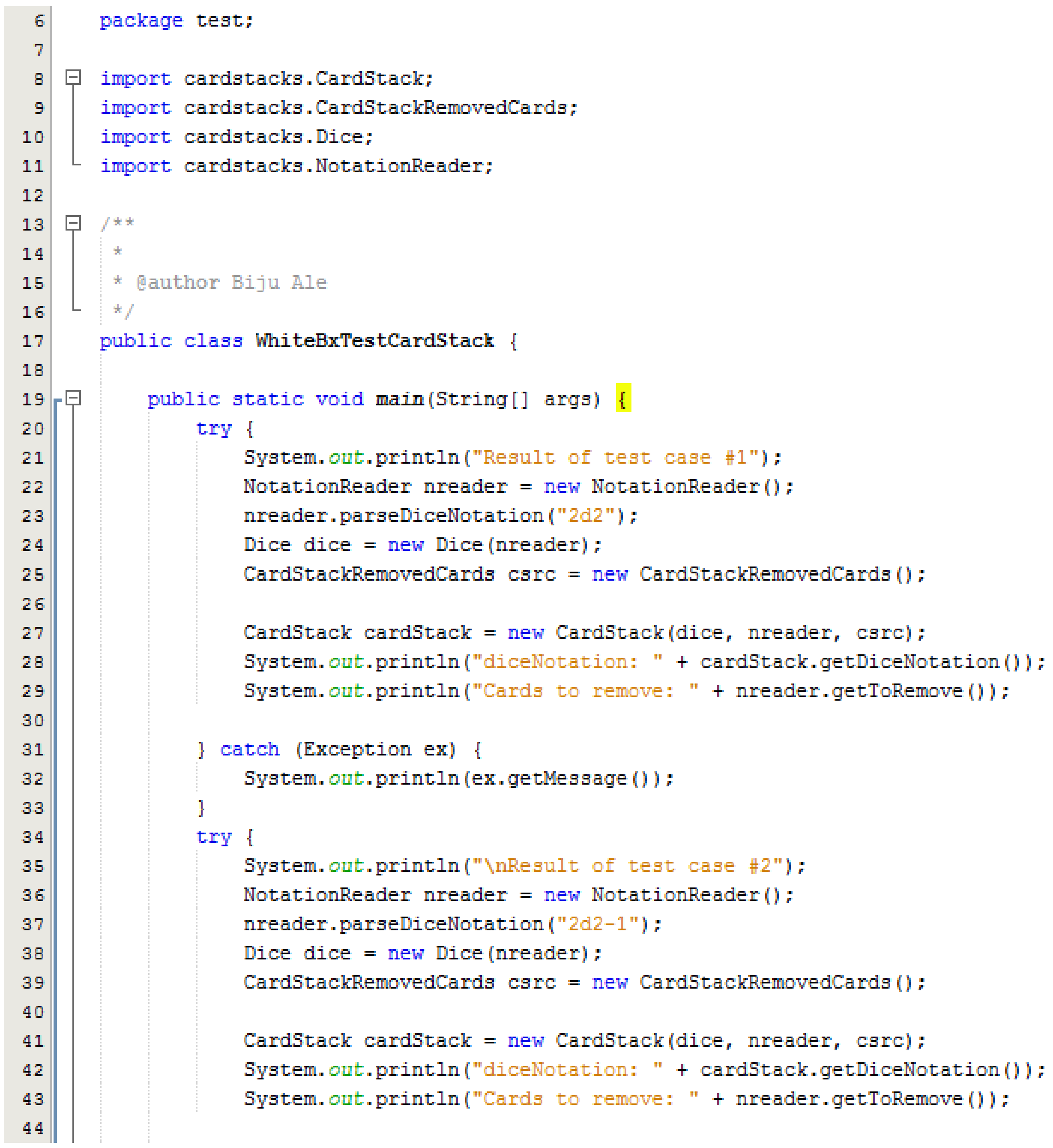
Method of test data selection:

- *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.

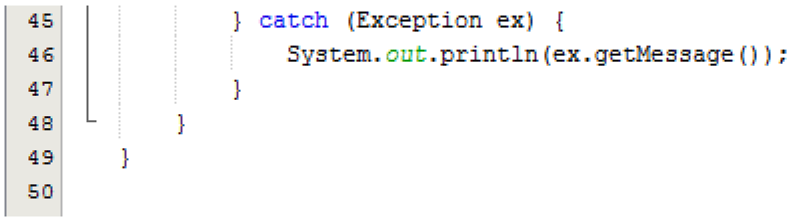
Test Execution

<u>Source Code</u>

[PLEASE TURN OVER]

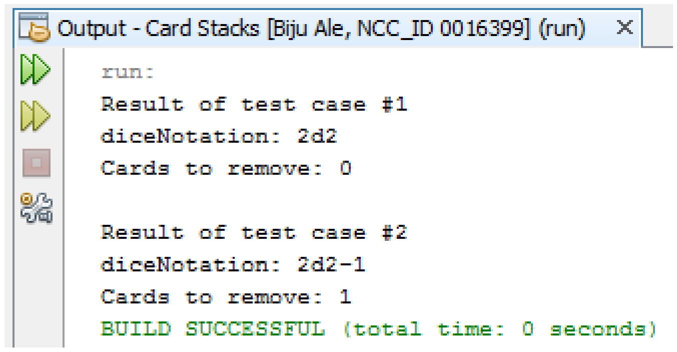


```java
package test;

import cardstacks.CardStack;
import cardstacks.CardStackRemovedCards;
import cardstacks.Dice;
import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class WhiteBxTestCardStack {

    public static void main(String[] args) {
        try {
            System.out.println("Result of test case #1");
            NotationReader nreader = new NotationReader();
            nreader.parseDiceNotation("2d2");
            Dice dice = new Dice(nreader);
            CardStackRemovedCards csrc = new CardStackRemovedCards();

            CardStack cardStack = new CardStack(dice, nreader, csrc);
            System.out.println("diceNotation: " + cardStack.getDiceNotation());
            System.out.println("Cards to remove: " + nreader.getToRemove());

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        try {
            System.out.println("\nResult of test case #2");
            NotationReader nreader = new NotationReader();
            nreader.parseDiceNotation("2d2-1");
            Dice dice = new Dice(nreader);
            CardStackRemovedCards csrc = new CardStackRemovedCards();

            CardStack cardStack = new CardStack(dice, nreader, csrc);
            System.out.println("diceNotation: " + cardStack.getDiceNotation());
            System.out.println("Cards to remove: " + nreader.getToRemove());

```



```
            } catch (Exception ex) {
                System.out.println(ex.getMessage());
            }
        }
    }

```

Output

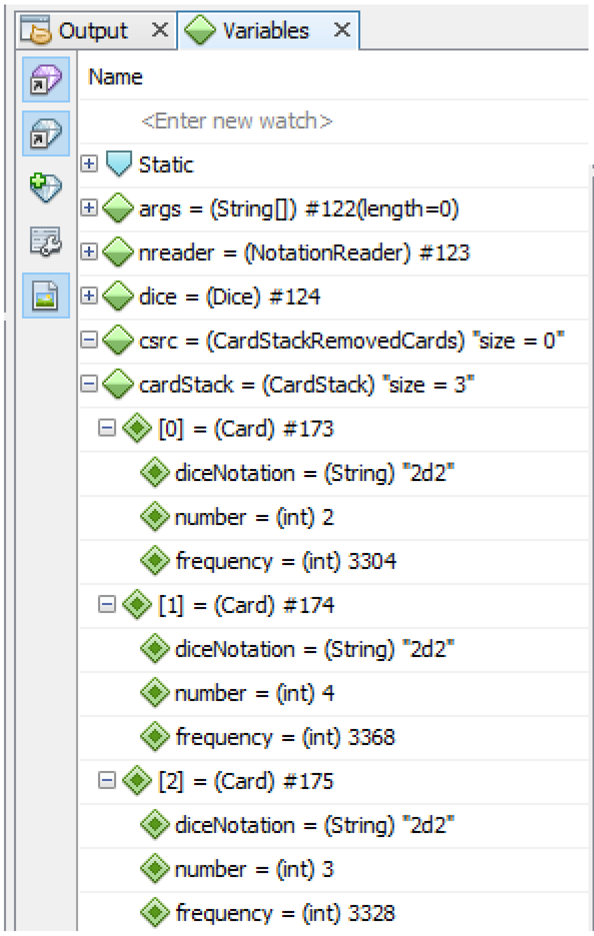


Output - Card Stacks [Biju Ale, NCC_ID 0016399] (run)

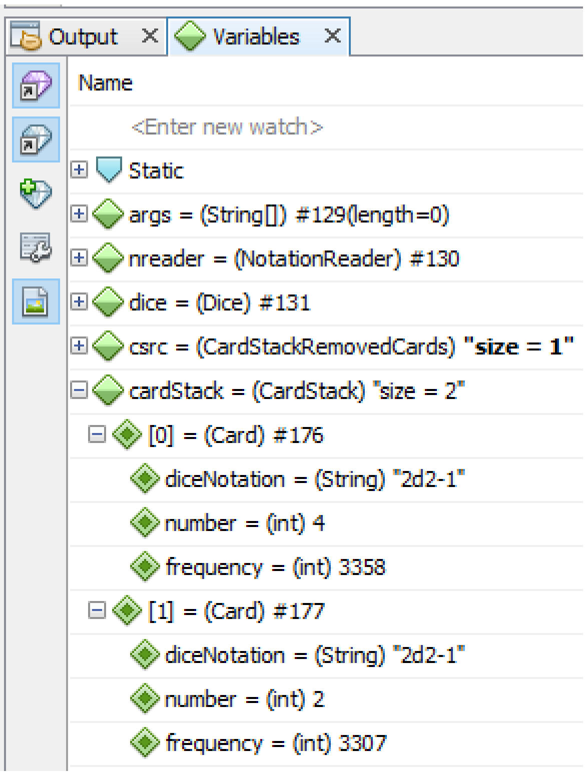
```
run:
Result of test case #1
diceNotation: 2d2
Cards to remove: 0

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:



```
Output  ×  Variables  ×
Name
    <Enter new watch>
⊞ Static
⊞ args = (String[]) #122(length=0)
⊞ nreader = (NotationReader) #123
⊞ dice = (Dice) #124
⊟ csrc = (CardStackRemovedCards) "size = 0"
⊟ cardStack = (CardStack) "size = 3"
  ⊟ [0] = (Card) #173
      diceNotation = (String) "2d2"
      number = (int) 2
      frequency = (int) 3304
  ⊟ [1] = (Card) #174
      diceNotation = (String) "2d2"
      number = (int) 4
      frequency = (int) 3368
  ⊟ [2] = (Card) #175
      diceNotation = (String) "2d2"
      number = (int) 3
      frequency = (int) 3328
```



```
Output  ×  Variables  ×
Name
    <Enter new watch>
⊞ Static
⊞ args = (String[]) #129(length=0)
⊞ nreader = (NotationReader) #130
⊞ dice = (Dice) #131
⊞ csrc = (CardStackRemovedCards) "size = 1"
⊟ cardStack = (CardStack) "size = 2"
  ⊟ [0] = (Card) #176
      diceNotation = (String) "2d2-1"
      number = (int) 4
      frequency = (int) 3358
  ⊟ [1] = (Card) #177
      diceNotation = (String) "2d2-1"
      number = (int) 2
      frequency = (int) 3307
```

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.

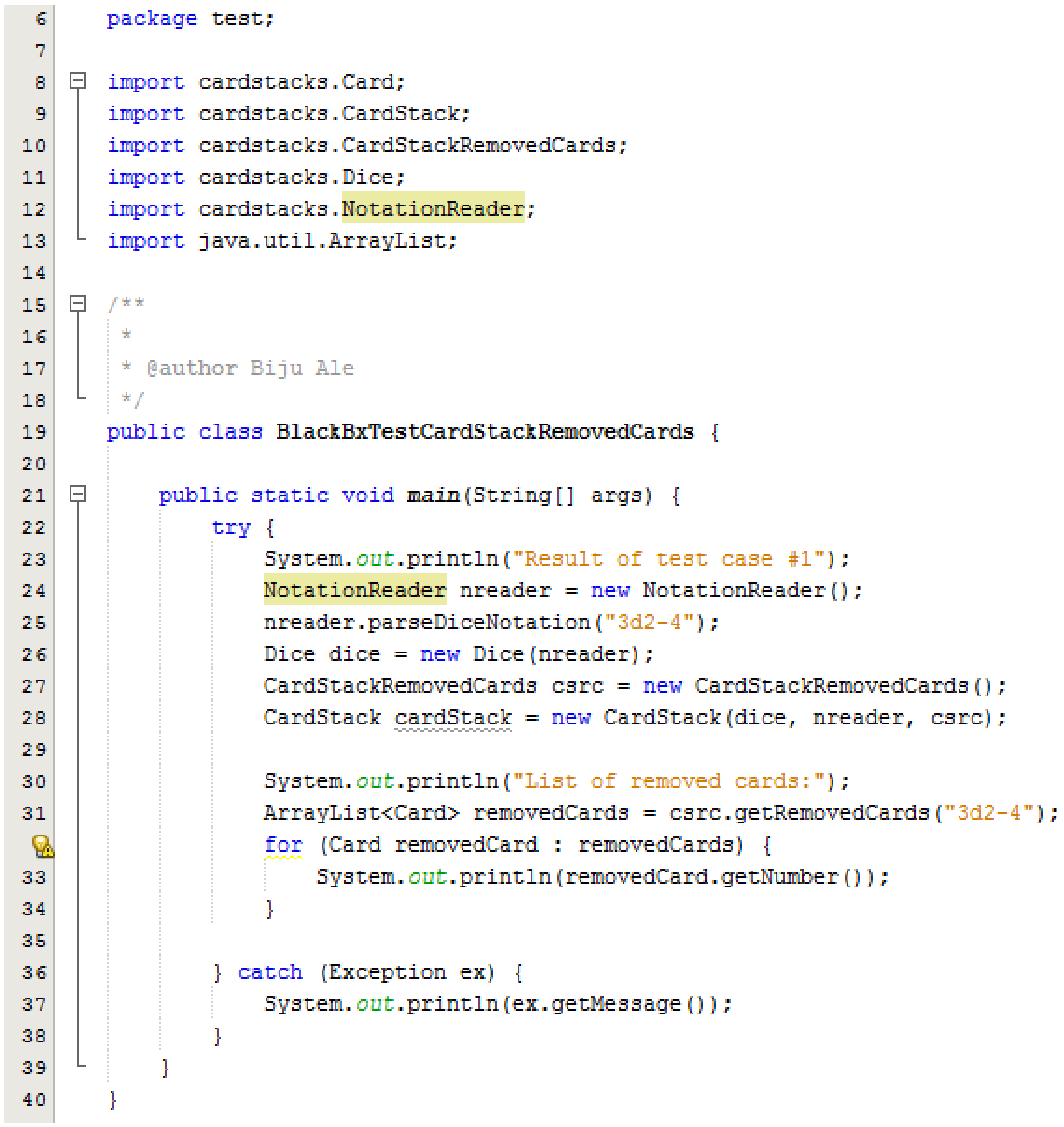
Method of test data selection:

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

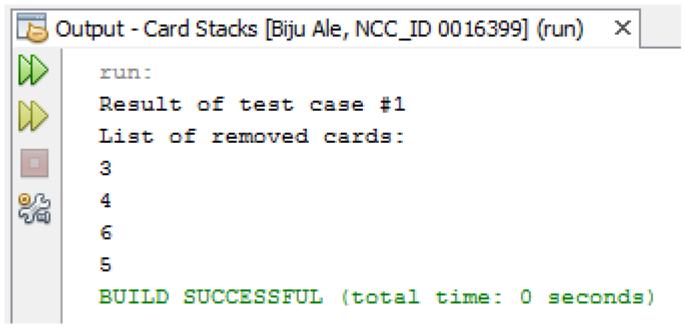
<u>Source Code</u>

[PLEASE TURN OVER]



```java
package test;

import cardstacks.Card;
import cardstacks.CardStack;
import cardstacks.CardStackRemovedCards;
import cardstacks.Dice;
import cardstacks.NotationReader;
import java.util.ArrayList;

/**
 *
 * @author Biju Ale
 */
public class BlackBxTestCardStackRemovedCards {

    public static void main(String[] args) {
        try {
            System.out.println("Result of test case #1");
            NotationReader nreader = new NotationReader();
            nreader.parseDiceNotation("3d2-4");
            Dice dice = new Dice(nreader);
            CardStackRemovedCards csrc = new CardStackRemovedCards();
            CardStack cardStack = new CardStack(dice, nreader, csrc);

            System.out.println("List of removed cards:");
            ArrayList<Card> removedCards = csrc.getRemovedCards("3d2-4");
            for (Card removedCard : removedCards) {
                System.out.println(removedCard.getNumber());
            }

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

<u>Output</u>



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.

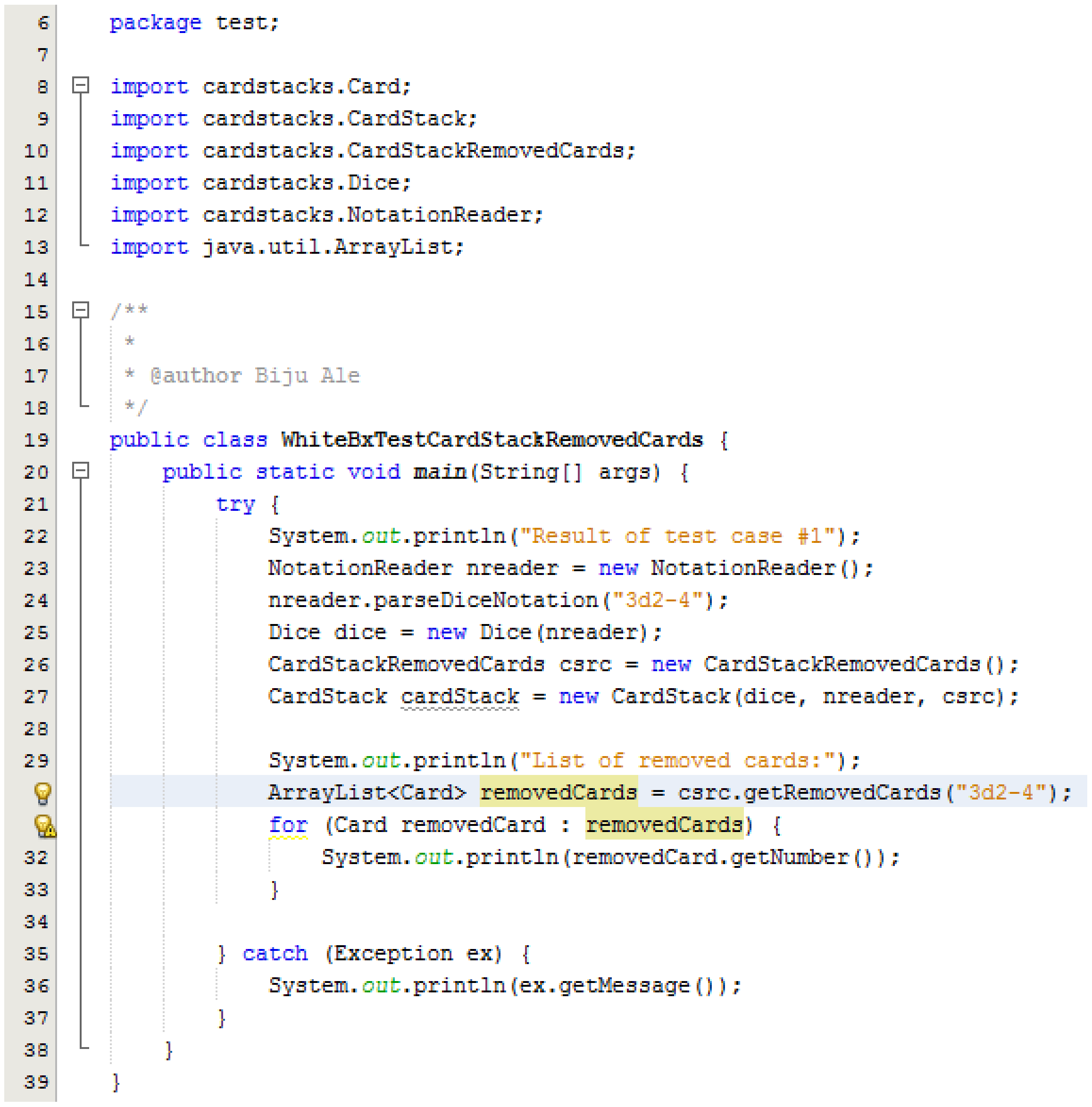
Method of test data selection:

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

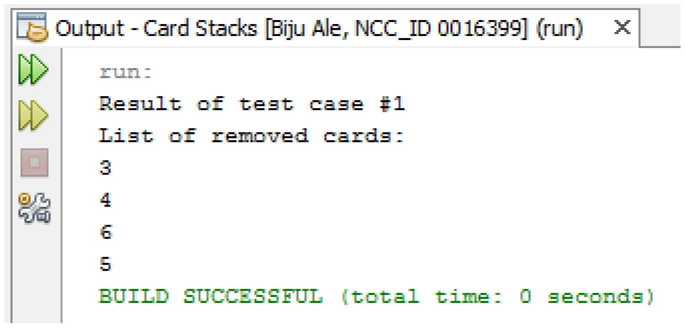
Source Code

[PLEASE TURN OVER]



```
package test;

import cardstacks.Card;
import cardstacks.CardStack;
import cardstacks.CardStackRemovedCards;
import cardstacks.Dice;
import cardstacks.NotationReader;
import java.util.ArrayList;

/**
 *
 * @author Biju Ale
 */
public class WhiteBxTestCardStackRemovedCards {
    public static void main(String[] args) {
        try {
            System.out.println("Result of test case #1");
            NotationReader nreader = new NotationReader();
            nreader.parseDiceNotation("3d2-4");
            Dice dice = new Dice(nreader);
            CardStackRemovedCards csrc = new CardStackRemovedCards();
            CardStack cardStack = new CardStack(dice, nreader, csrc);

            System.out.println("List of removed cards:");
            ArrayList<Card> removedCards = csrc.getRemovedCards("3d2-4");
            for (Card removedCard : removedCards) {
                System.out.println(removedCard.getNumber());
            }

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

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 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

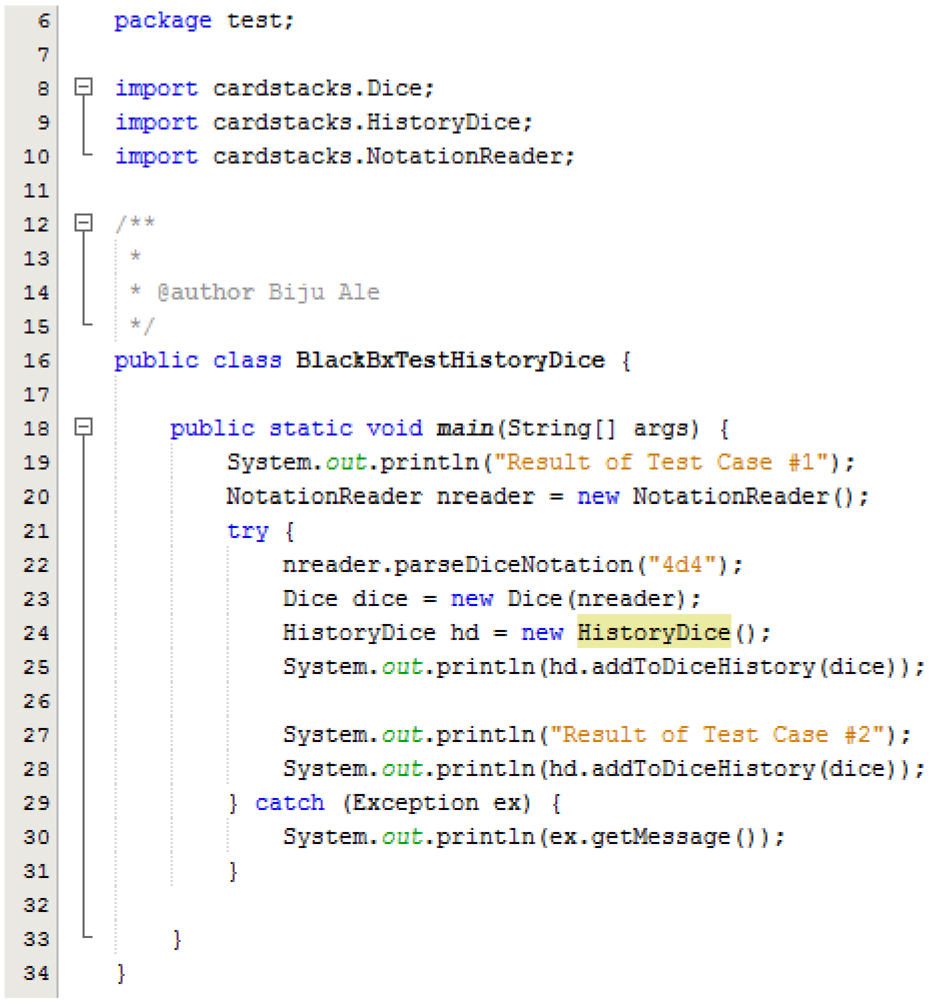
#	Method	Test Data	Input type	Expected Outcome
1	boolean addToDiceHistory(Dice dice)	dice	Valid object	Returns TRUE after adding the dice to history.
2	boolean addToDiceHistory(Dice)	dice	Invalid object (duplicate dice)	Returns FALSE after matching dice to its history.

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.

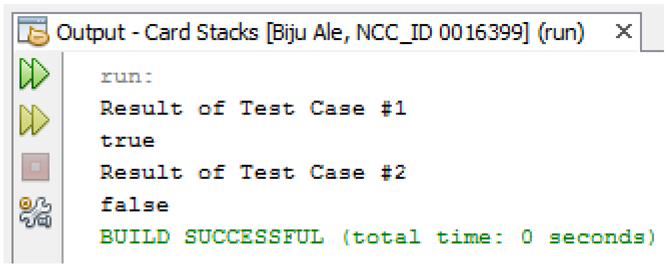
Test Execution

<u>Source Code</u>



```
package test;

import cardstacks.Dice;
import cardstacks.HistoryDice;
import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class BlackBxTestHistoryDice {

    public static void main(String[] args) {
        System.out.println("Result of Test Case #1");
        NotationReader nreader = new NotationReader();
        try {
            nreader.parseDiceNotation("4d4");
            Dice dice = new Dice(nreader);
            HistoryDice hd = new HistoryDice();
            System.out.println(hd.addToDiceHistory(dice));

            System.out.println("Result of Test Case #2");
            System.out.println(hd.addToDiceHistory(dice));
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

    }
}
```

Output



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

```
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

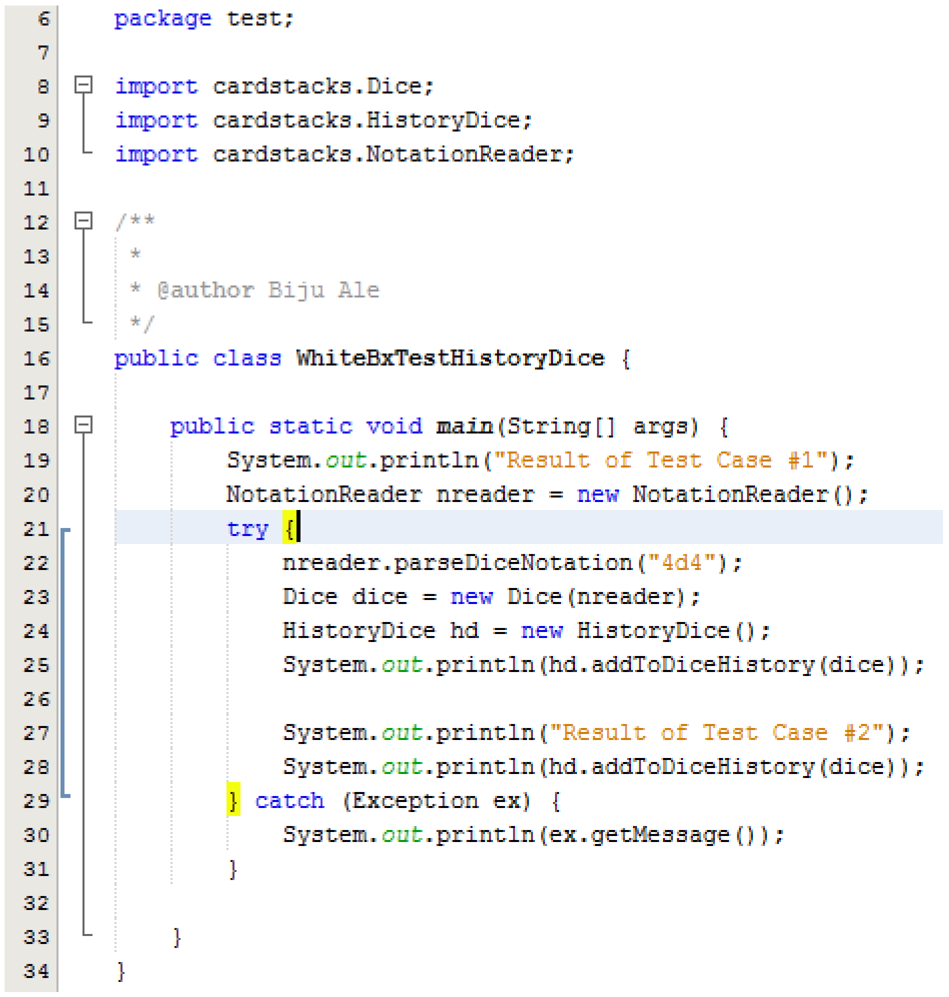
#	Method	Test Data	Input type	Expected Outcome
1	boolean addToDiceHistory(Dice dice)	dice	Valid object	Returns TRUE after adding the dice to history.
2	boolean addToDiceHistory(Dice)	dice	Invalid object (duplicate dice)	Returns FALSE after matching dice to its history.

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.

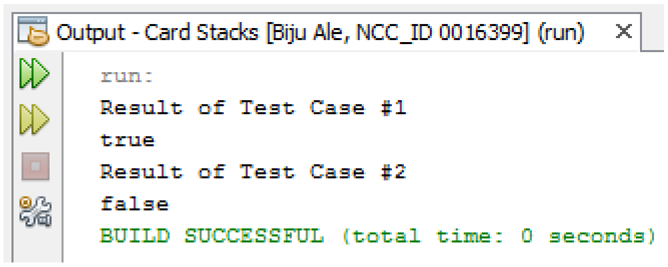
Test Execution

Source Code



```java
package test;

import cardstacks.Dice;
import cardstacks.HistoryDice;
import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class WhiteBxTestHistoryDice {

    public static void main(String[] args) {
        System.out.println("Result of Test Case #1");
        NotationReader nreader = new NotationReader();
        try {
            nreader.parseDiceNotation("4d4");
            Dice dice = new Dice(nreader);
            HistoryDice hd = new HistoryDice();
            System.out.println(hd.addToDiceHistory(dice));

            System.out.println("Result of Test Case #2");
            System.out.println(hd.addToDiceHistory(dice));
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

    }
}
```

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

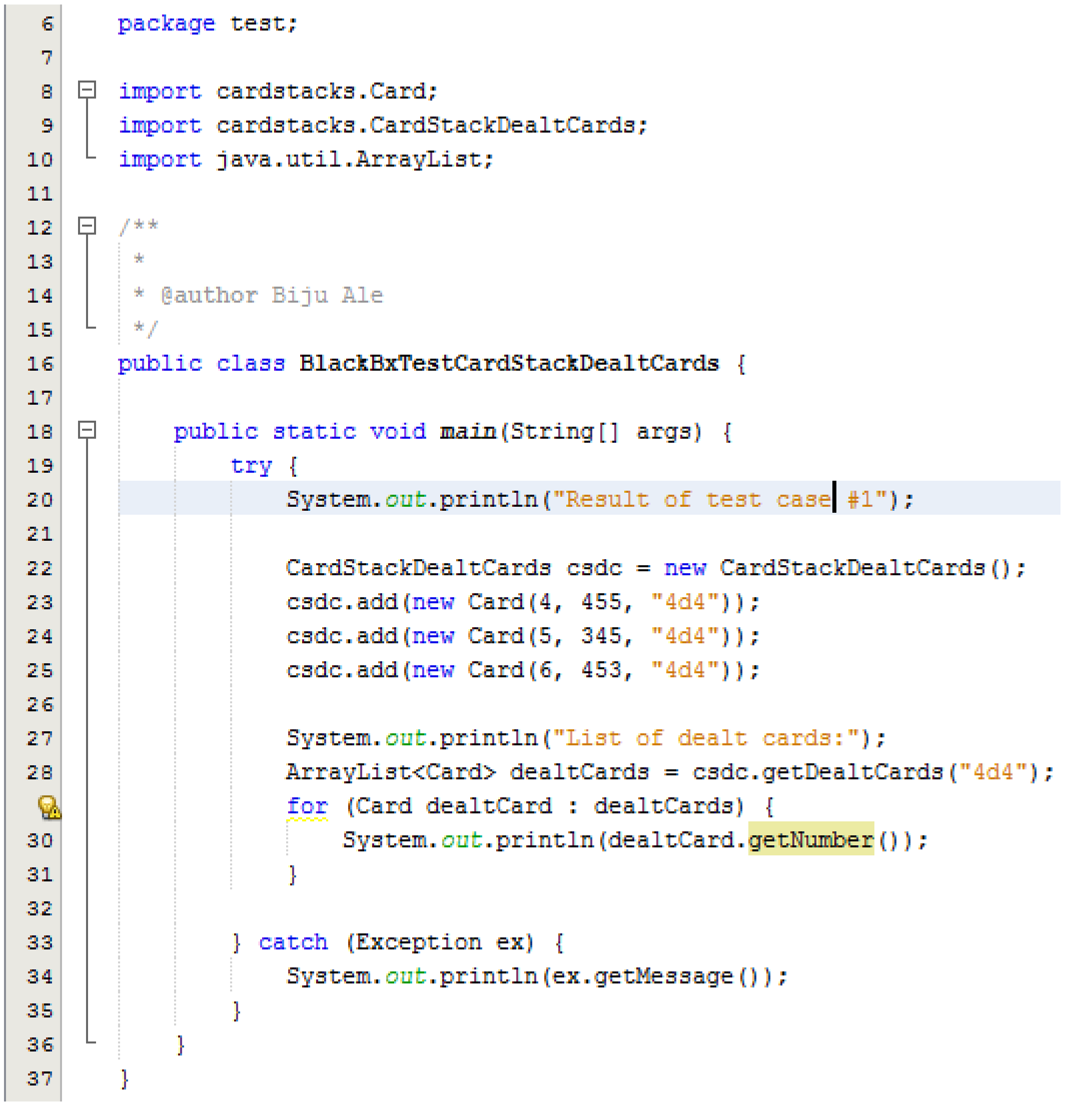
#	Method	Test Data	Input type	Expected Outcome
1	ArrayList<Card> getDealtCards(String diceNotation)	“4d4”	Valid string	Returns correct no. i.e. of dealt cards. i.e. 4 dealt cards.

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.

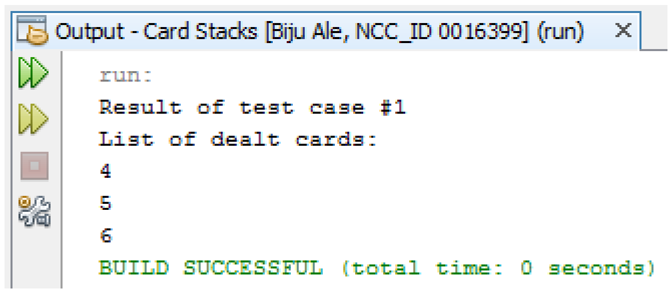
Test Execution

Source Code



```
package test;

import cardstacks.Card;
import cardstacks.CardStackDealtCards;
import java.util.ArrayList;

/**
 *
 * @author Biju Ale
 */
public class BlackBxTestCardStackDealtCards {

    public static void main(String[] args) {
        try {
            System.out.println("Result of test case #1");

            CardStackDealtCards csdc = new CardStackDealtCards();
            csdc.add(new Card(4, 455, "4d4"));
            csdc.add(new Card(5, 345, "4d4"));
            csdc.add(new Card(6, 453, "4d4"));

            System.out.println("List of dealt cards:");
            ArrayList<Card> dealtCards = csdc.getDealtCards("4d4");
            for (Card dealtCard : dealtCards) {
                System.out.println(dealtCard.getNumber());
            }

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

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

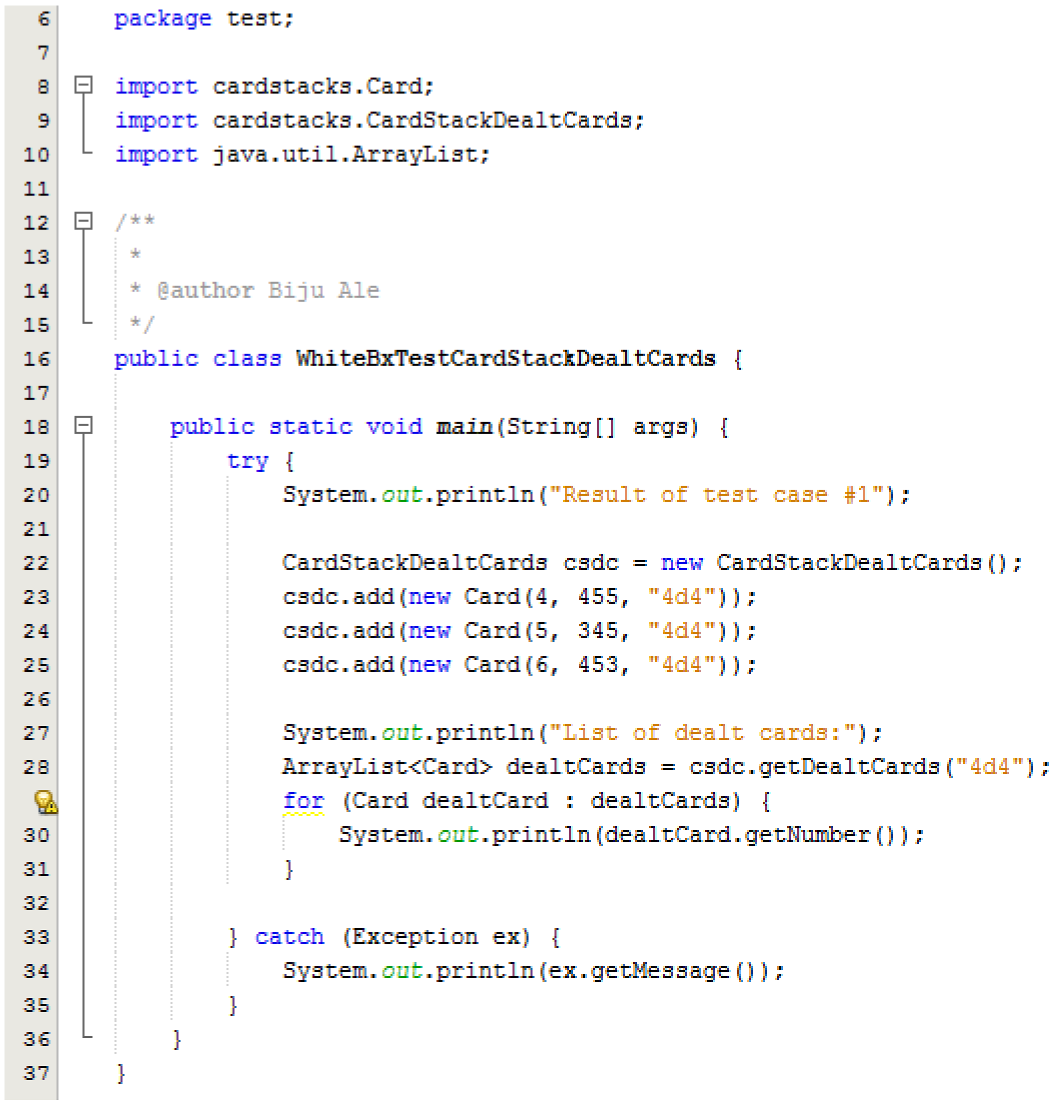
#	Method	Test Data	Input type	Expected Outcome
1	ArrayList<Card> getDealtCards(String diceNotation)	“4d4”	Valid string	Returns correct no. i.e. of dealt cards. i.e. 4 dealt cards.

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.

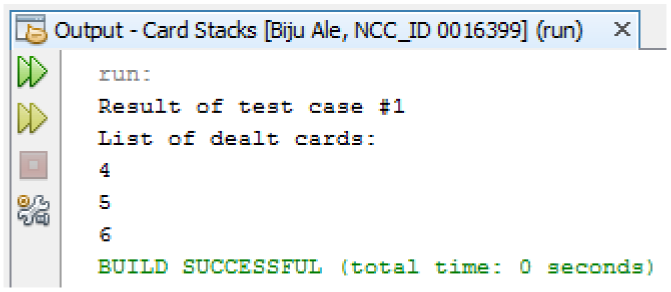
Test Execution

Source Code



```java
package test;

import cardstacks.Card;
import cardstacks.CardStackDealtCards;
import java.util.ArrayList;

/**
 *
 * @author Biju Ale
 */
public class WhiteBxTestCardStackDealtCards {

    public static void main(String[] args) {
        try {
            System.out.println("Result of test case #1");

            CardStackDealtCards csdc = new CardStackDealtCards();
            csdc.add(new Card(4, 455, "4d4"));
            csdc.add(new Card(5, 345, "4d4"));
            csdc.add(new Card(6, 453, "4d4"));

            System.out.println("List of dealt cards:");
            ArrayList<Card> dealtCards = csdc.getDealtCards("4d4");
            for (Card dealtCard : dealtCards) {
                System.out.println(dealtCard.getNumber());
            }

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

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.

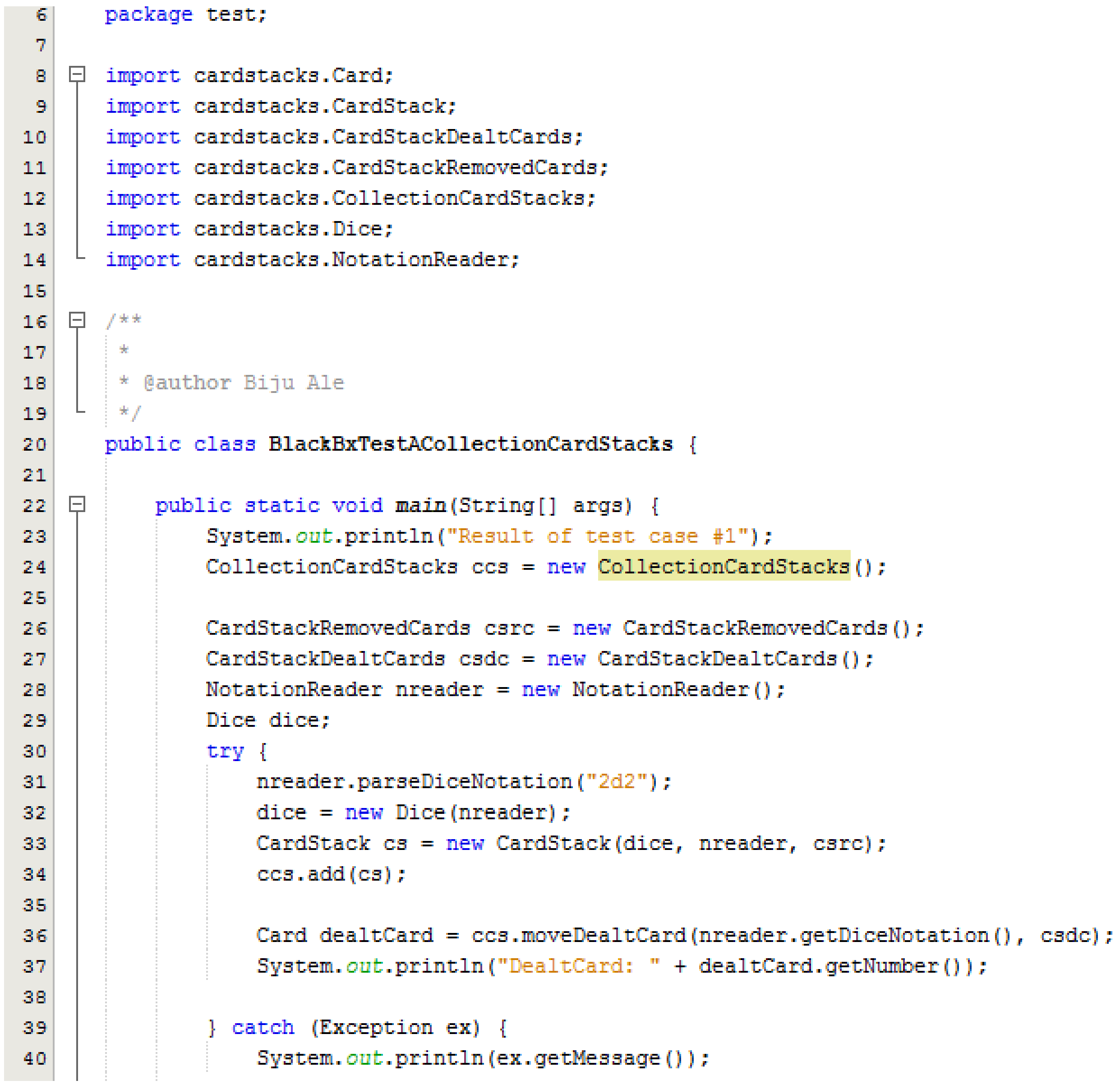
Method of test data selection:

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

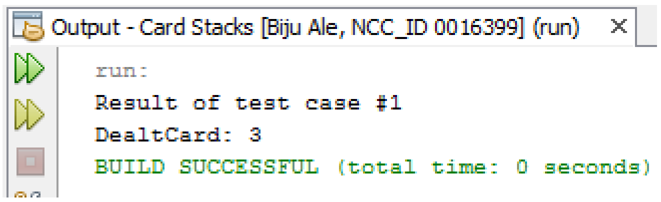
Source Code

[PLEASE TURN OVER]



```java
package test;

import cardstacks.Card;
import cardstacks.CardStack;
import cardstacks.CardStackDealtCards;
import cardstacks.CardStackRemovedCards;
import cardstacks.CollectionCardStacks;
import cardstacks.Dice;
import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class BlackBxTestACollectionCardStacks {

    public static void main(String[] args) {
        System.out.println("Result of test case #1");
        CollectionCardStacks ccs = new CollectionCardStacks();

        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;
        try {
            nreader.parseDiceNotation("2d2");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);

            Card dealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
            System.out.println("DealtCard: " + dealtCard.getNumber());

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
```

Output



```
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.

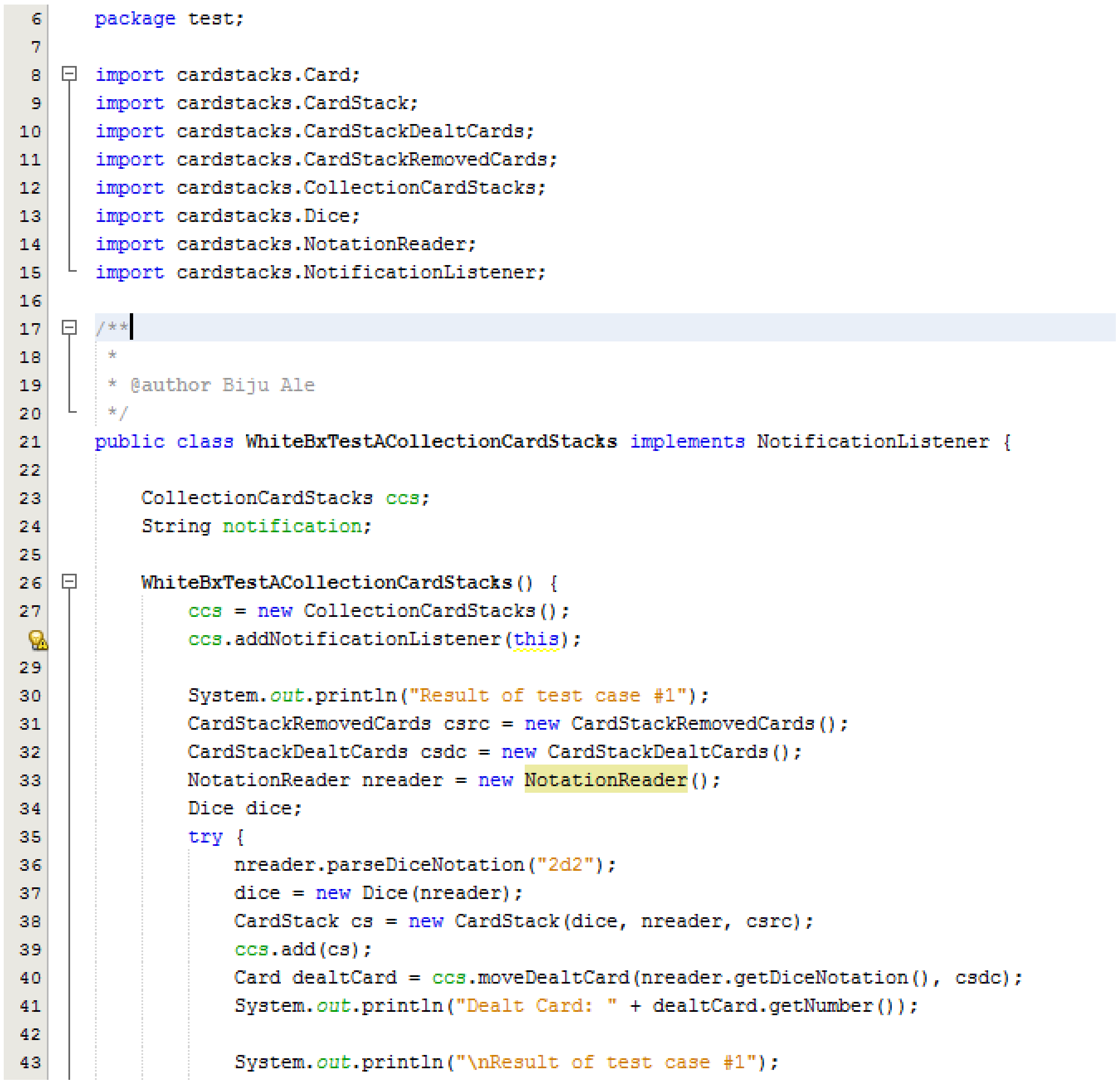
Method of test data selection:

- 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.

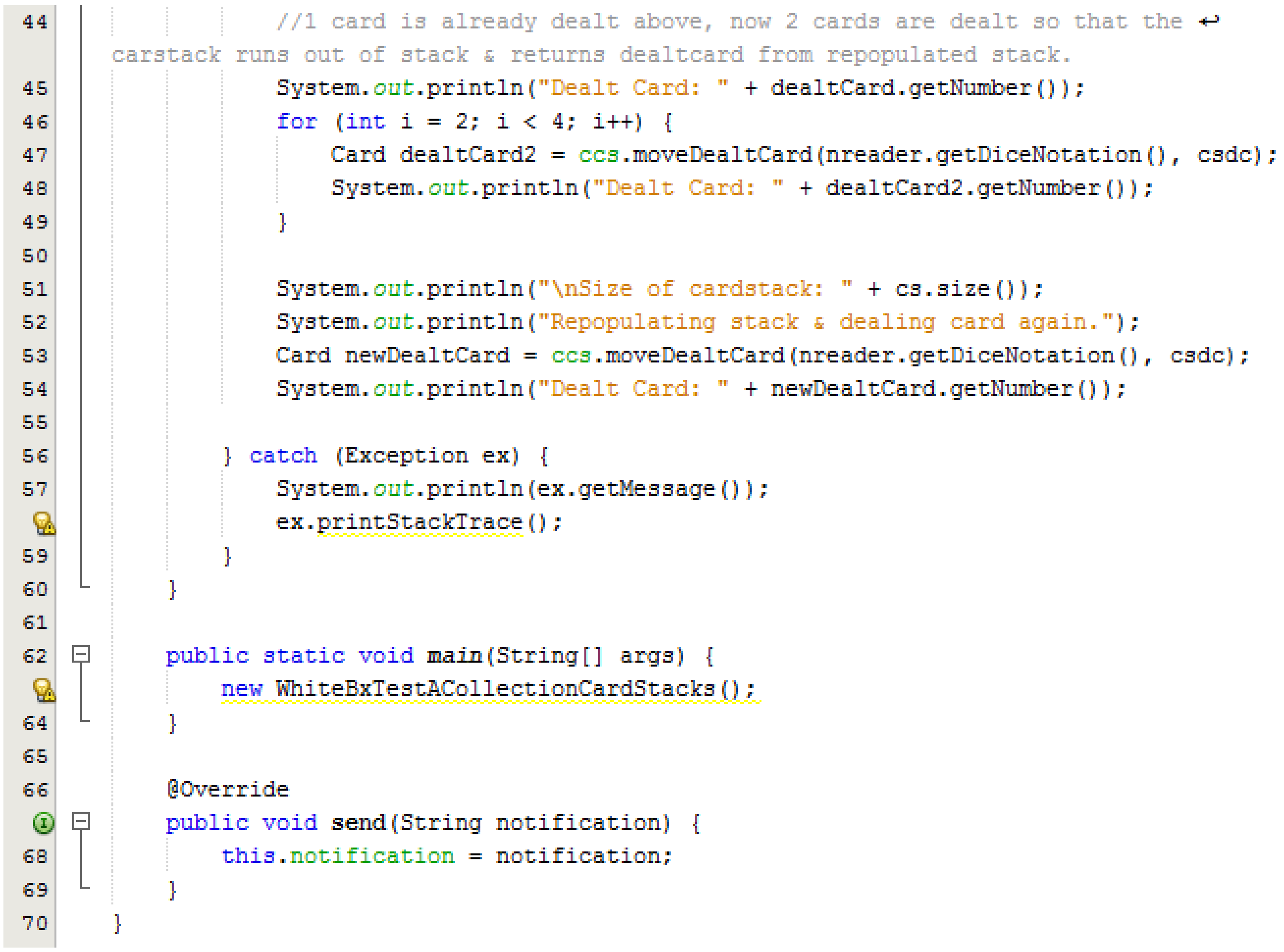
Test Execution

Source Code

[PLEASE TURN OVER]

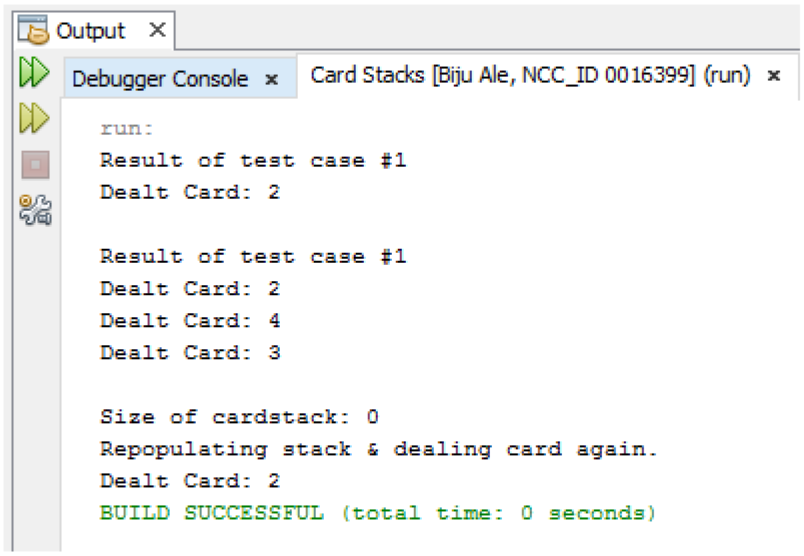


```java
package test;

import cardstacks.Card;
import cardstacks.CardStack;
import cardstacks.CardStackDealtCards;
import cardstacks.CardStackRemovedCards;
import cardstacks.CollectionCardStacks;
import cardstacks.Dice;
import cardstacks.NotationReader;
import cardstacks.NotificationListener;

/**
 *
 * @author Biju Ale
 */
public class WhiteBxTestACollectionCardStacks implements NotificationListener {

    CollectionCardStacks ccs;
    String notification;

    WhiteBxTestACollectionCardStacks() {
        ccs = new CollectionCardStacks();
        ccs.addNotificationListener(this);

        System.out.println("Result of test case #1");
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;
        try {
            nreader.parseDiceNotation("2d2");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);
            Card dealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
            System.out.println("Dealt Card: " + dealtCard.getNumber());

            System.out.println("\nResult of test case #1");
```



```
                //1 card is already dealt above, now 2 cards are dealt so that the carstack runs out of stack & returns dealtcard from repopulated stack.
                System.out.println("Dealt Card: " + dealtCard.getNumber());
                for (int i = 2; i < 4; i++) {
                    Card dealtCard2 = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
                    System.out.println("Dealt Card: " + dealtCard2.getNumber());
                }

                System.out.println("\nSize of cardstack: " + cs.size());
                System.out.println("Repopulating stack & dealing card again.");
                Card newDealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
                System.out.println("Dealt Card: " + newDealtCard.getNumber());

            } catch (Exception ex) {
                System.out.println(ex.getMessage());
                ex.printStackTrace();
            }
        }

    public static void main(String[] args) {
        new WhiteBxTestACollectionCardStacks();
    }

    @Override
    public void send(String notification) {
        this.notification = notification;
    }
}
```

Output



```
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

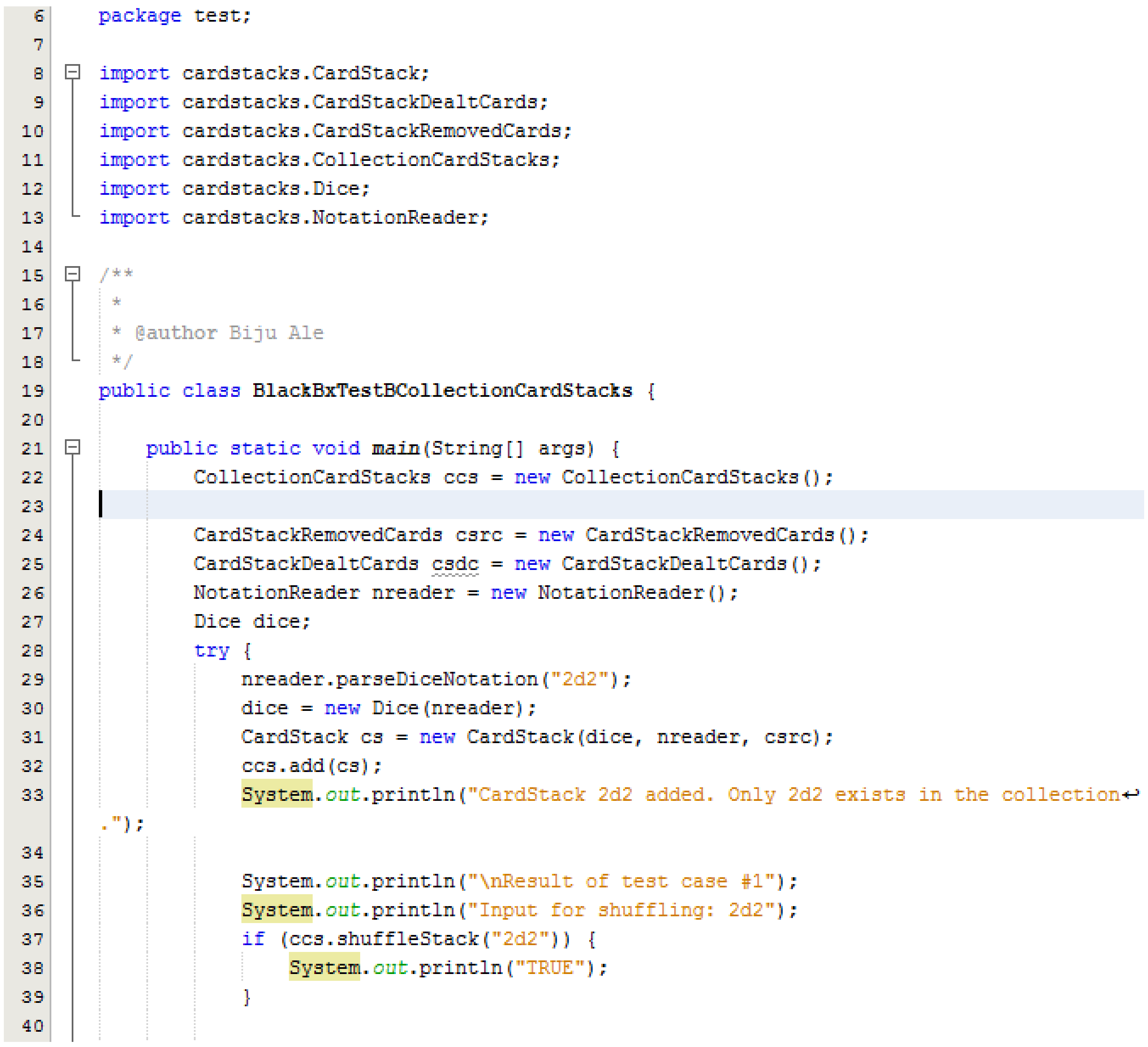
Testingclass: cardstacks.NotationReader
Testingtype: Black Box / Unit Testing

#	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.

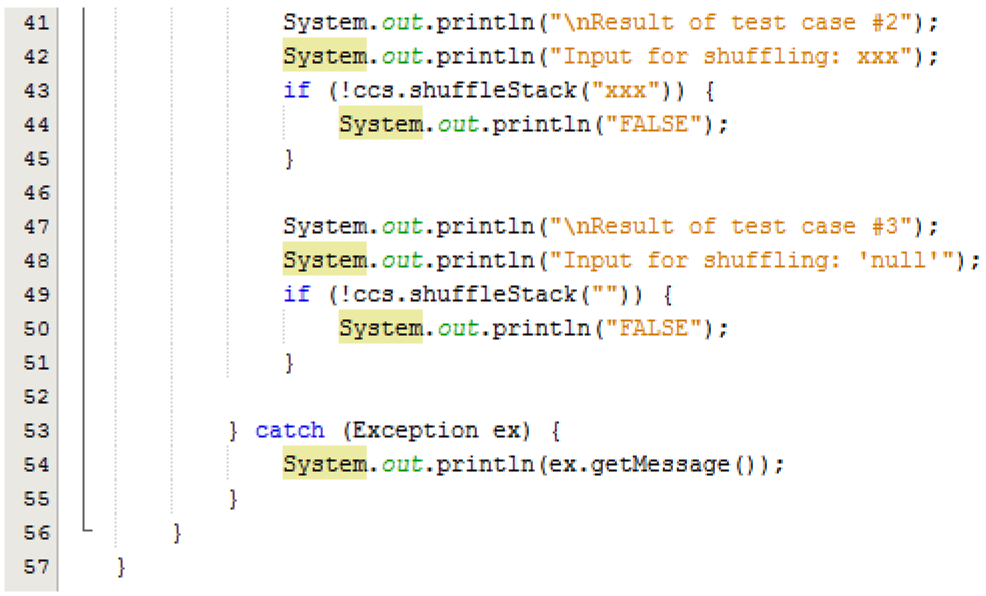
Test Execution

<u>Source Code</u>

[PLEASE TURN OVER]

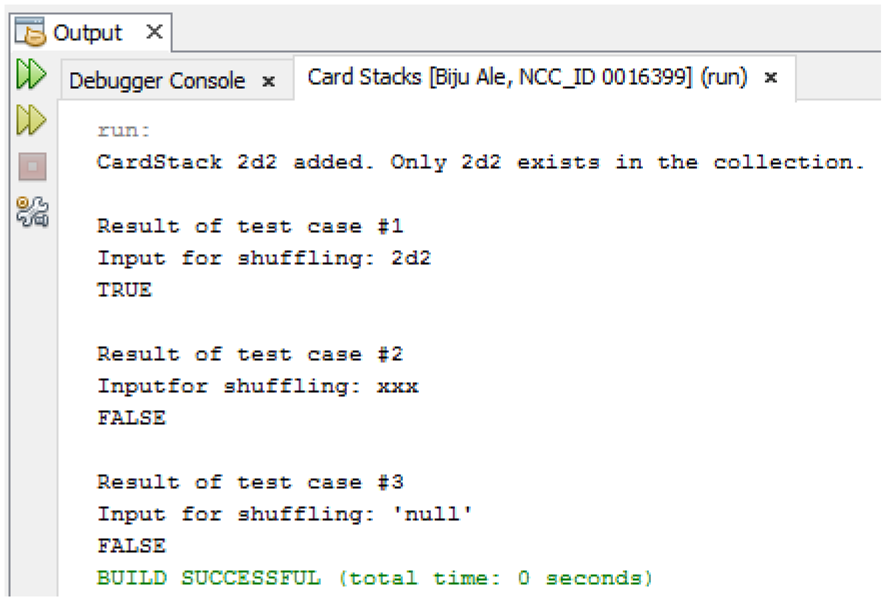


```java
package test;

import cardstacks.CardStack;
import cardstacks.CardStackDealtCards;
import cardstacks.CardStackRemovedCards;
import cardstacks.CollectionCardStacks;
import cardstacks.Dice;
import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class BlackBxTestBCollectionCardStacks {

    public static void main(String[] args) {
        CollectionCardStacks ccs = new CollectionCardStacks();

        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;
        try {
            nreader.parseDiceNotation("2d2");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);
            System.out.println("CardStack 2d2 added. Only 2d2 exists in the collection.");

            System.out.println("\nResult of test case #1");
            System.out.println("Input for shuffling: 2d2");
            if (ccs.shuffleStack("2d2")) {
                System.out.println("TRUE");
            }

```



```java
                System.out.println("\nResult of test case #2");
                System.out.println("Input for shuffling: xxx");
                if (!ccs.shuffleStack("xxx")) {
                    System.out.println("FALSE");
                }

                System.out.println("\nResult of test case #3");
                System.out.println("Input for shuffling: 'null'");
                if (!ccs.shuffleStack("")) {
                    System.out.println("FALSE");
                }

            } catch (Exception ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
```

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.

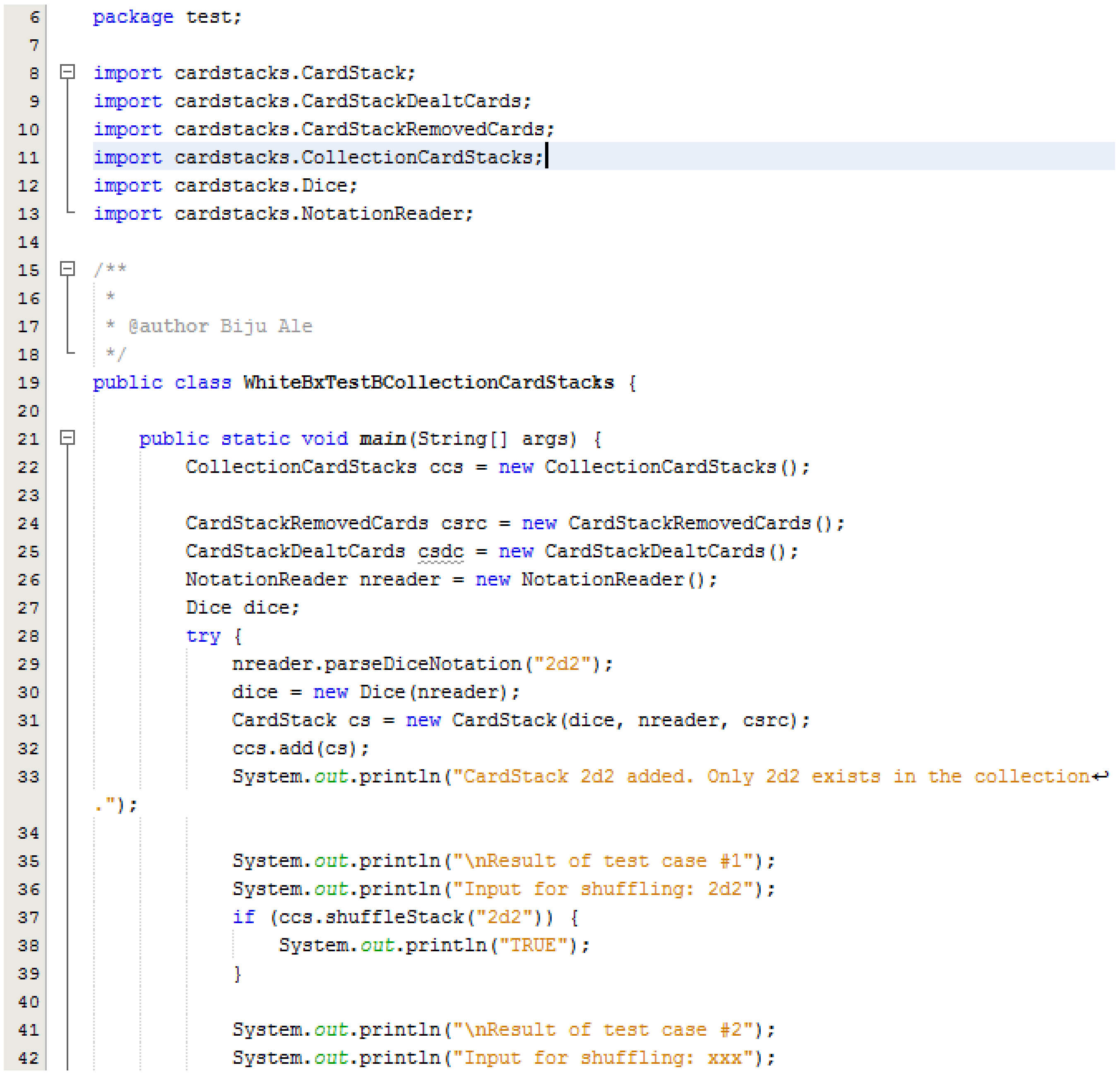
Test Suite No.18

Testingclass: cardstacks.NotationReader
Testingtype: White Box / Unit Testing

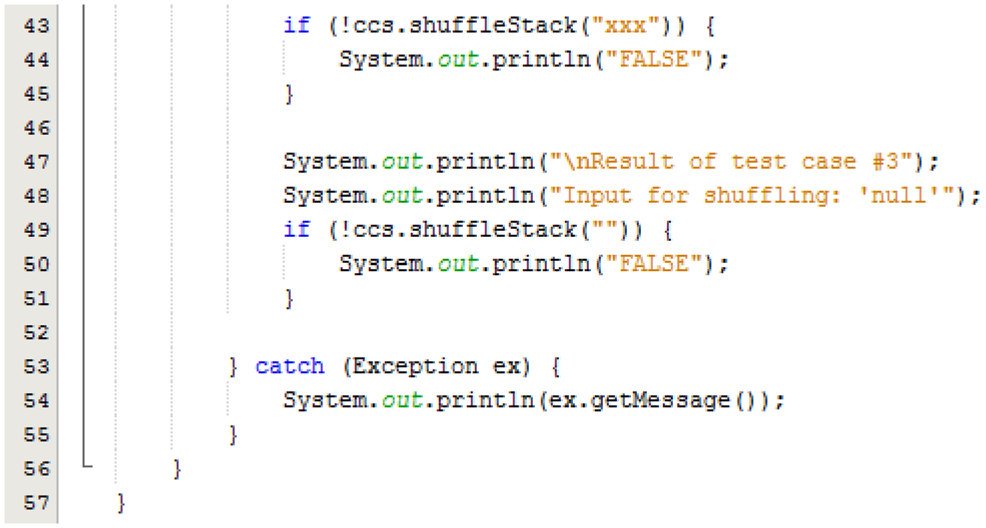
#	Method	Test Data	Input type	Expected Outcome
1	boolean shuffleStack(String diceNotation)	“4d4”	Valid string	Returns true after changing order of 4d4 cardstack.
2	boolean shuffleStack(String diceNotation)	“xxx”	Valid string	Returns false.
3	boolean shuffleStack(String diceNotation)	“”	Null	Returns false.

Test Execution

<u>Source Code</u>

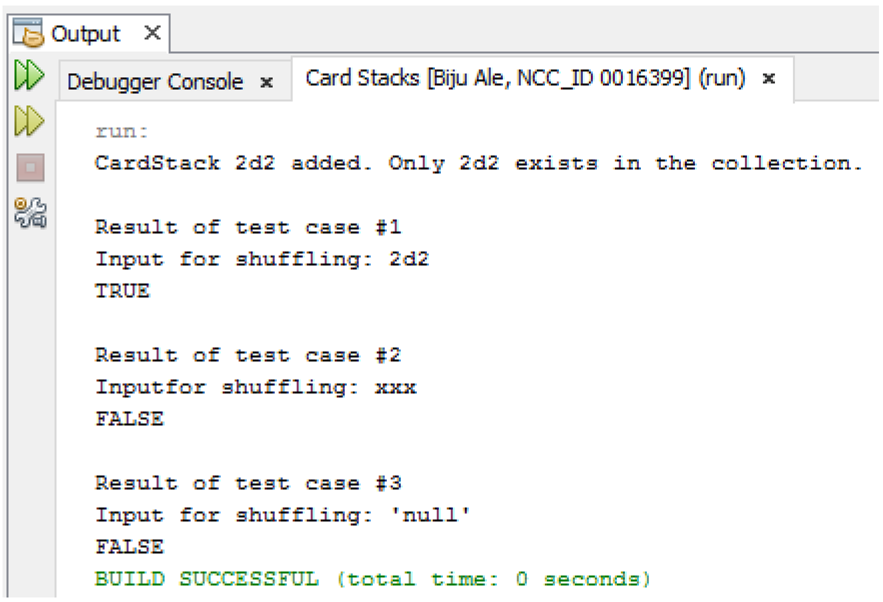


```java
package test;

import cardstacks.CardStack;
import cardstacks.CardStackDealtCards;
import cardstacks.CardStackRemovedCards;
import cardstacks.CollectionCardStacks;
import cardstacks.Dice;
import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class WhiteBxTestBCollectionCardStacks {

    public static void main(String[] args) {
        CollectionCardStacks ccs = new CollectionCardStacks();

        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;
        try {
            nreader.parseDiceNotation("2d2");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);
            System.out.println("CardStack 2d2 added. Only 2d2 exists in the collection.");

            System.out.println("\nResult of test case #1");
            System.out.println("Input for shuffling: 2d2");
            if (ccs.shuffleStack("2d2")) {
                System.out.println("TRUE");
            }

            System.out.println("\nResult of test case #2");
            System.out.println("Input for shuffling: xxx");
```



```
                if (!ccs.shuffleStack("xxx")) {
                    System.out.println("FALSE");
                }

                System.out.println("\nResult of test case #3");
                System.out.println("Input for shuffling: 'null'");
                if (!ccs.shuffleStack("")) {
                    System.out.println("FALSE");
                }

            } catch (Exception ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
```

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

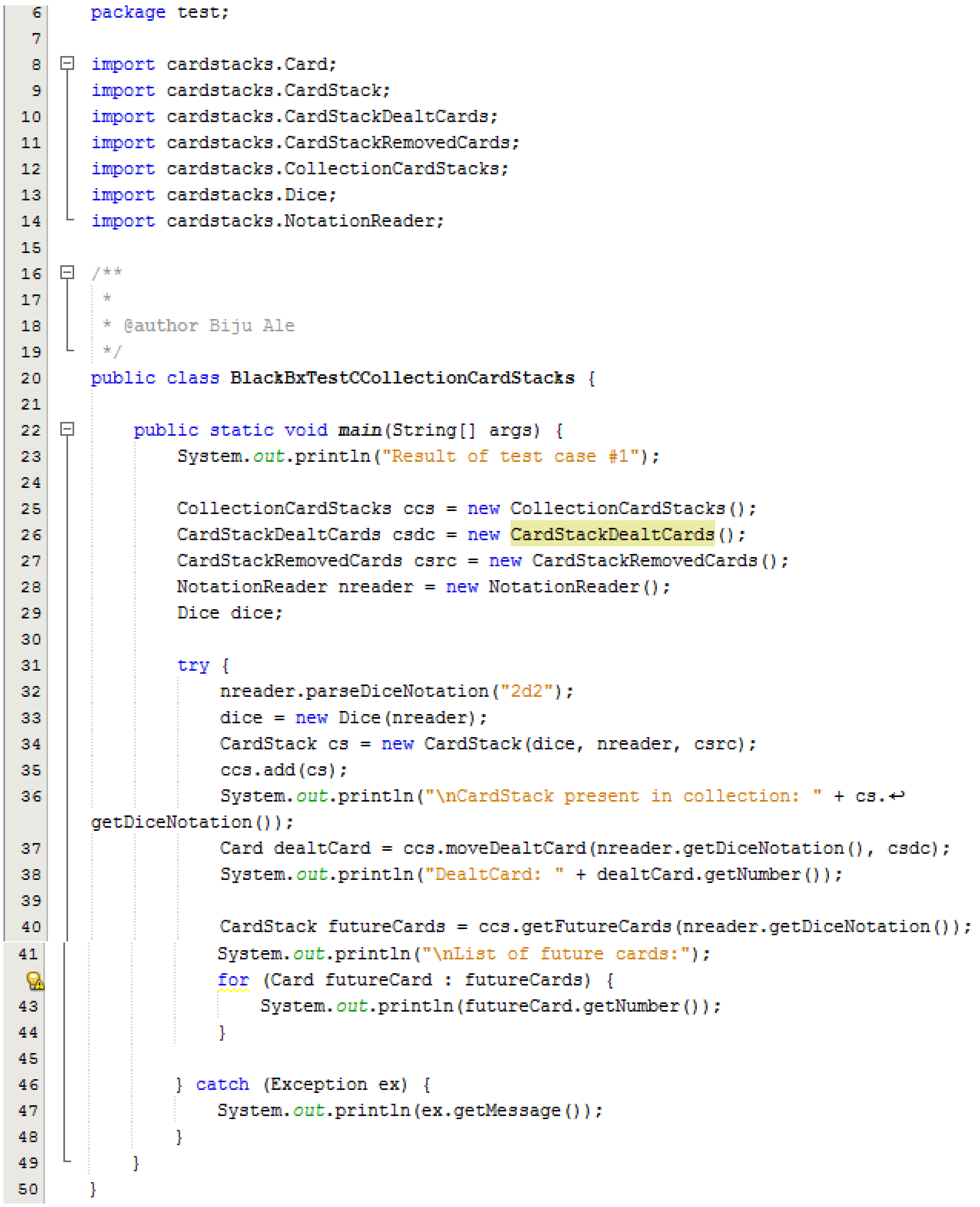
Testingclass: cardstacks.NotationReader
Testingtype: Black Box / Unit Testing

#	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

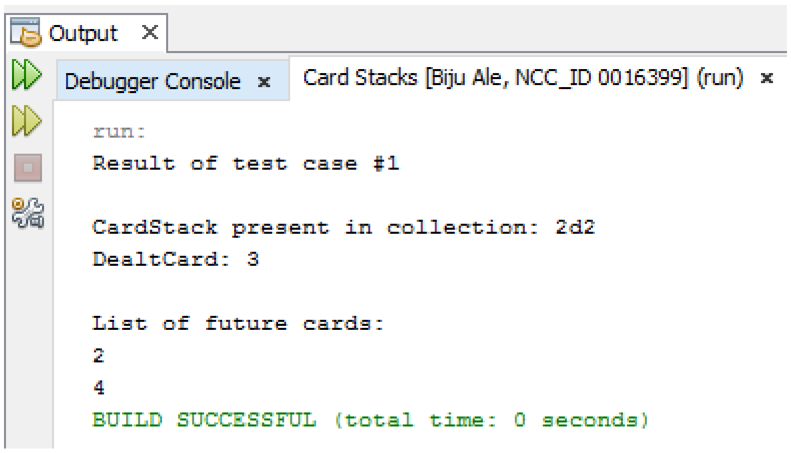
Source Code

[PLEASE TURN OVER]



```java
package test;

import cardstacks.Card;
import cardstacks.CardStack;
import cardstacks.CardStackDealtCards;
import cardstacks.CardStackRemovedCards;
import cardstacks.CollectionCardStacks;
import cardstacks.Dice;
import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class BlackBxTestCCollectionCardStacks {

    public static void main(String[] args) {
        System.out.println("Result of test case #1");

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackDealtCards csdc = new CardStackDealtCards();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("2d2");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);
            System.out.println("\nCardStack present in collection: " + cs.getDiceNotation());
            Card dealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
            System.out.println("DealtCard: " + dealtCard.getNumber());

            CardStack futureCards = ccs.getFutureCards(nreader.getDiceNotation());
            System.out.println("\nList of future cards:");
            for (Card futureCard : futureCards) {
                System.out.println(futureCard.getNumber());
            }

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

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 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

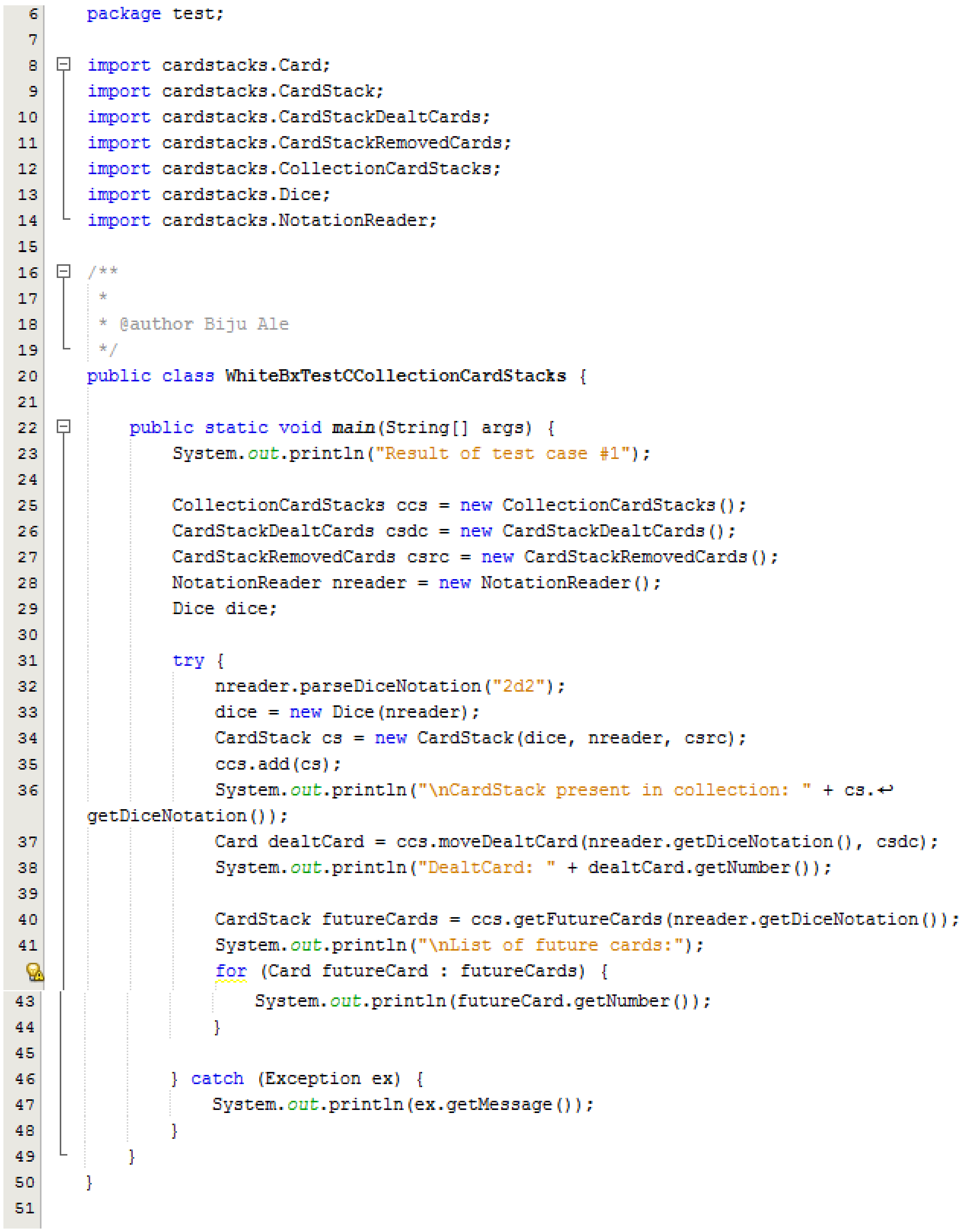
Testingclass: cardstacks.NotationReader
Testingtype: White Box / Unit Testing

#	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

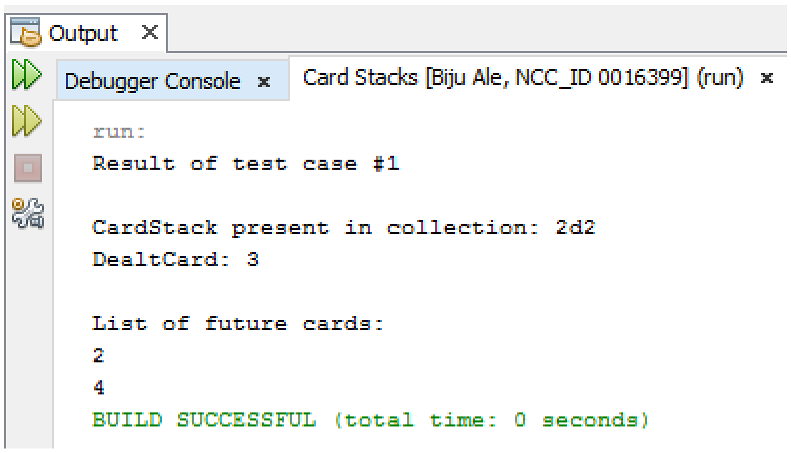
<u>Source Code</u>

[PLEASE TURN OVER]



```java
package test;

import cardstacks.Card;
import cardstacks.CardStack;
import cardstacks.CardStackDealtCards;
import cardstacks.CardStackRemovedCards;
import cardstacks.CollectionCardStacks;
import cardstacks.Dice;
import cardstacks.NotationReader;

/**
 *
 * @author Biju Ale
 */
public class WhiteBxTestCCollectionCardStacks {

    public static void main(String[] args) {
        System.out.println("Result of test case #1");

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackDealtCards csdc = new CardStackDealtCards();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("2d2");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);
            System.out.println("\nCardStack present in collection: " + cs.getDiceNotation());
            Card dealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);
            System.out.println("DealtCard: " + dealtCard.getNumber());

            CardStack futureCards = ccs.getFutureCards(nreader.getDiceNotation());
            System.out.println("\nList of future cards:");
            for (Card futureCard : futureCards) {
                System.out.println(futureCard.getNumber());
            }

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}

```

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.

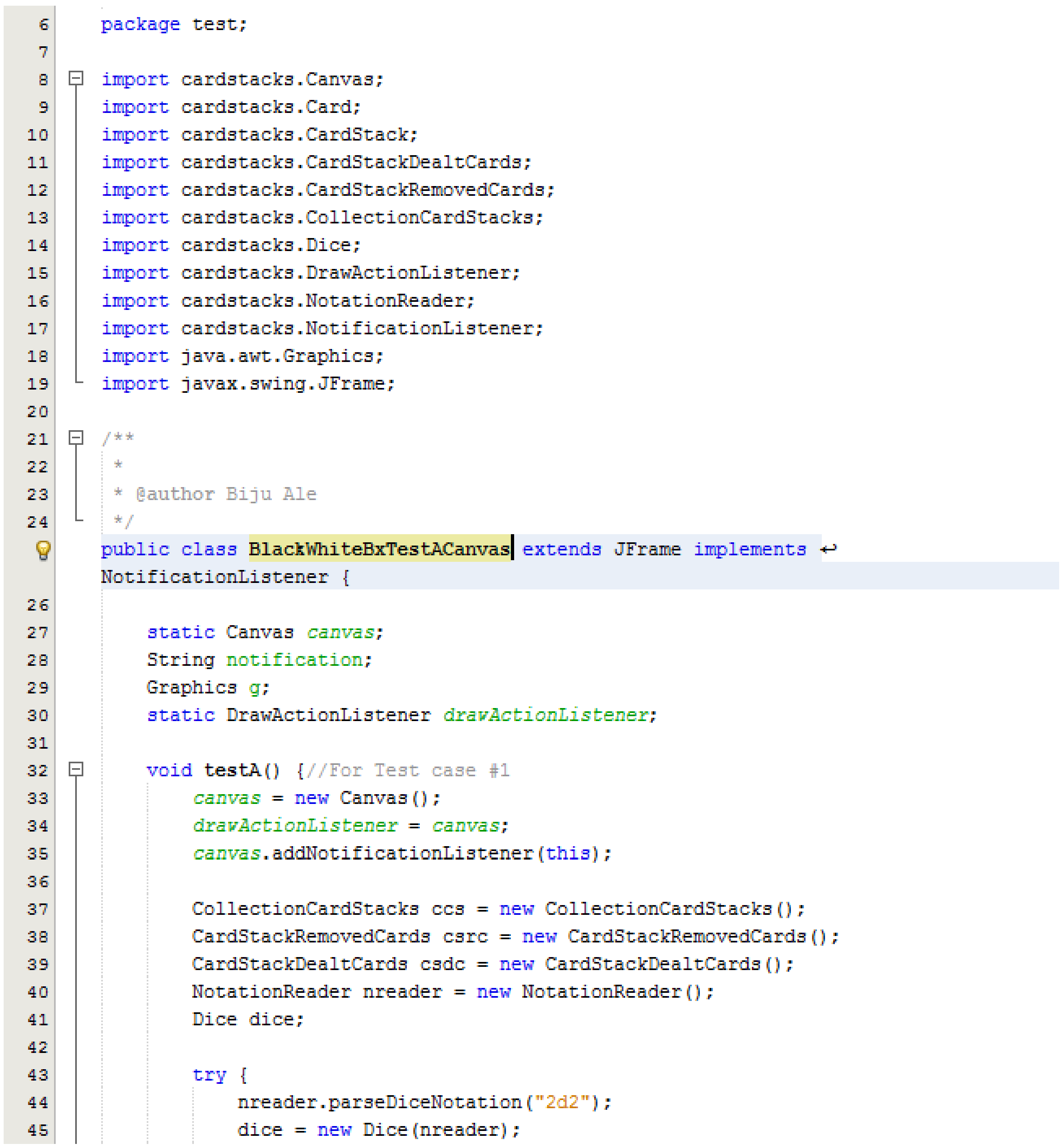
Test Suite No.21

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”

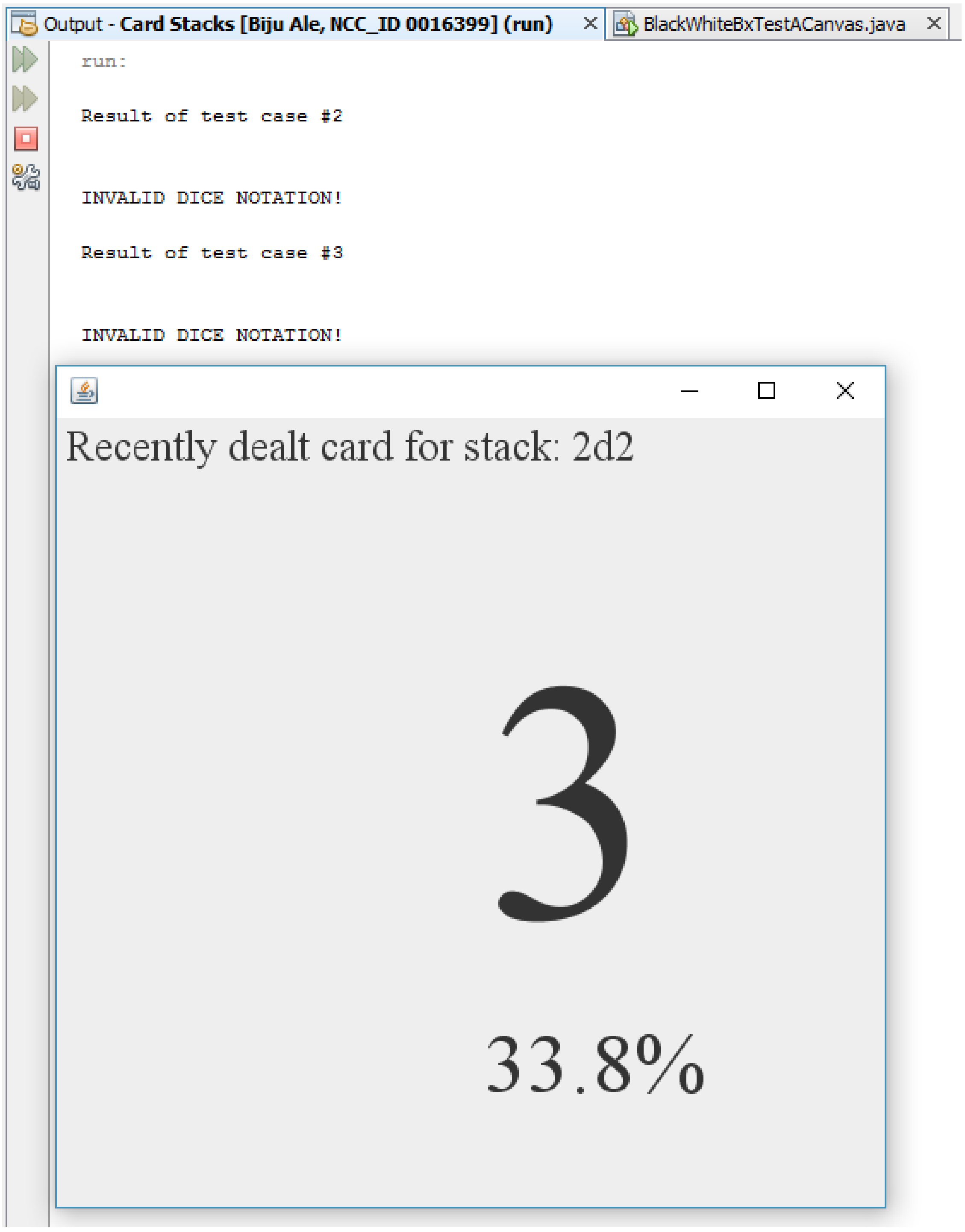
Test Execution

Source Code



```java
package test;

import cardstacks.Canvas;
import cardstacks.Card;
import cardstacks.CardStack;
import cardstacks.CardStackDealtCards;
import cardstacks.CardStackRemovedCards;
import cardstacks.CollectionCardStacks;
import cardstacks.Dice;
import cardstacks.DrawActionListener;
import cardstacks.NotationReader;
import cardstacks.NotificationListener;
import java.awt.Graphics;
import javax.swing.JFrame;

/**
 *
 * @author Biju Ale
 */
public class BlackWhiteBxTestACanvas extends JFrame implements NotificationListener {

    static Canvas canvas;
    String notification;
    Graphics g;
    static DrawActionListener drawActionListener;

    void testA() {//For Test case #1
        canvas = new Canvas();
        drawActionListener = canvas;
        canvas.addNotificationListener(this);

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("2d2");
            dice = new Dice(nreader);
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);
            Card singlDealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), ↩
csdc);

            canvas.sendSingleDealtCard(singlDealtCard);
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_GET_CARD;

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    void testB() {//For Test case #2
        System.out.println("\nResult of test case #2");
        canvas = new Canvas();
        drawActionListener = canvas;
        canvas.addNotificationListener(this);

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("xxx");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);
            Card singlDealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), ↩
csdc);

            canvas.sendSingleDealtCard(singlDealtCard);
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_GET_CARD;

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    void testC() {//For Test case #3
        System.out.println("\nResult of test case #3");
        canvas = new Canvas();
        drawActionListener = canvas;
        canvas.addNotificationListener(this);

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);
            Card singlDealtCard = ccs.moveDealtCard(nreader.getDiceNotation(), csdc);

            canvas.sendSingleDealtCard(singlDealtCard);
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_GET_CARD;

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    public static void main(String[] args) {
        BlackWhiteBxTestACanvas btc = new BlackWhiteBxTestACanvas();
        btc.setSize(500, 500);
        btc.setVisible(true);

        btc.testA();
        btc.add(canvas);

        btc.testB();

        btc.testC();

    }

    @Override
    public void send(String notification) {
        this.notification = notification;
    }
}

```

Output


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

BlackWhiteBxTestACanvas.java

```
run:

Result of test case #2


INVALID DICE NOTATION!

Result of test case #3


INVALID DICE NOTATION!
```

Recently dealt card for stack: 2d2

3

33.8%

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.

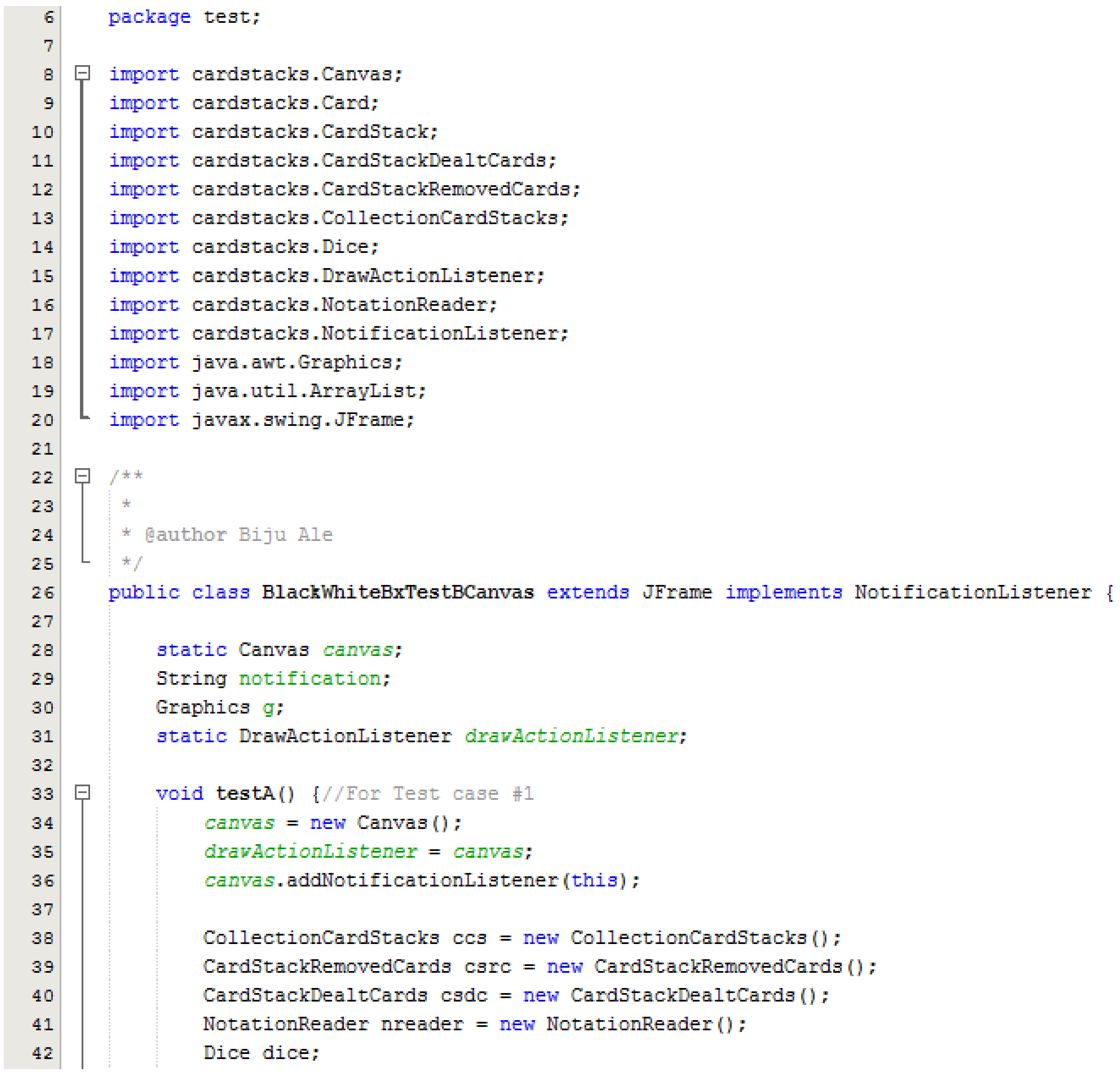
Test Suite No.22

Testingclass: cardstacks.Canvas
Testingtype: Black Box & White Box / Unit Testing

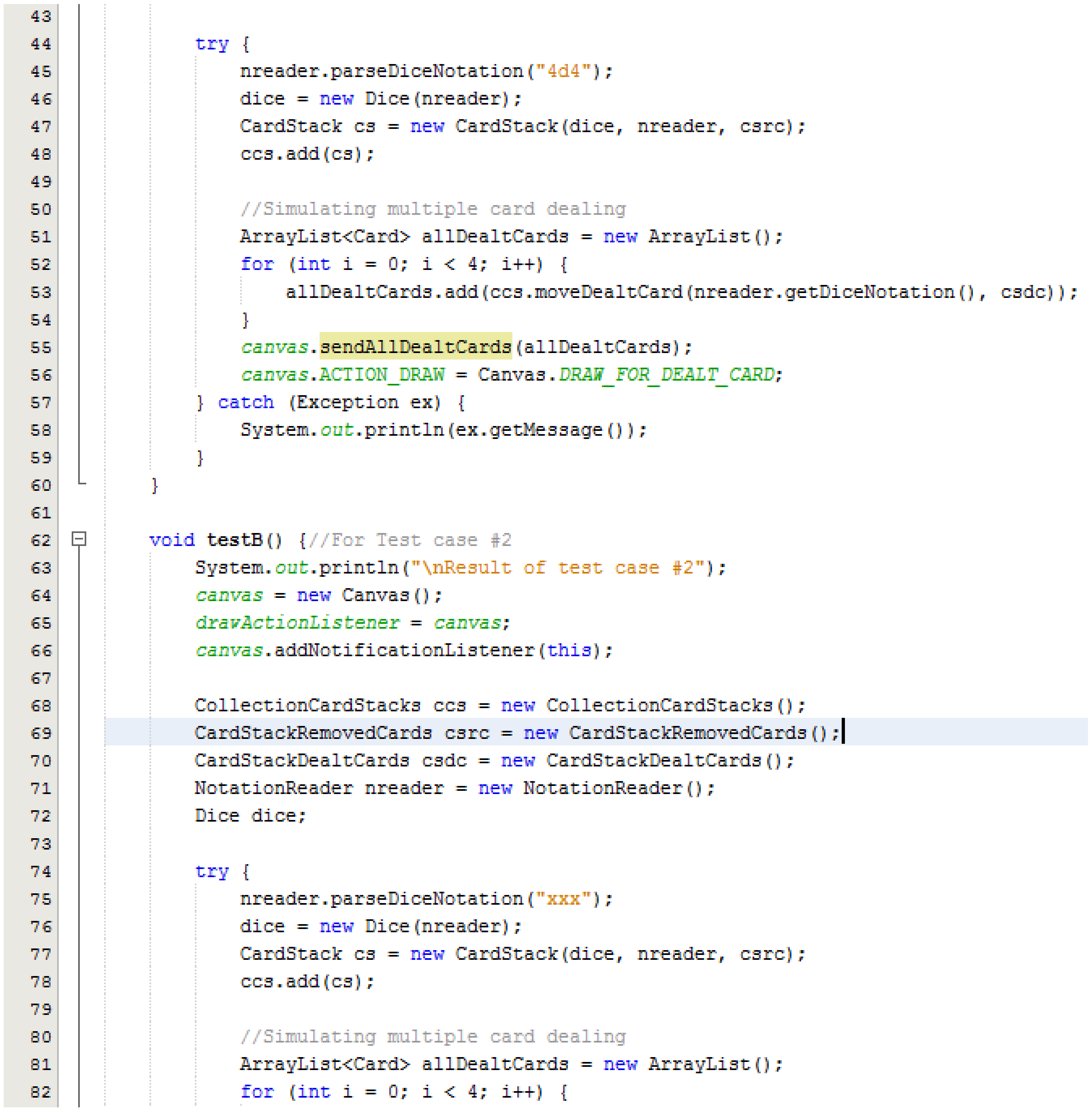
#	Method	Test Data	Input type	Expected Outcome
1	sendAllDealtCards(ArrayList allDealtCards)	allDealtCards	Valid object	Draws Graph of all dealt cards on JPanel
2	sendAllDealtCards(ArrayList allDealtCards)	allDealtCards	Null (object has null data member i.e. null dice notation)	Throws exception with message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

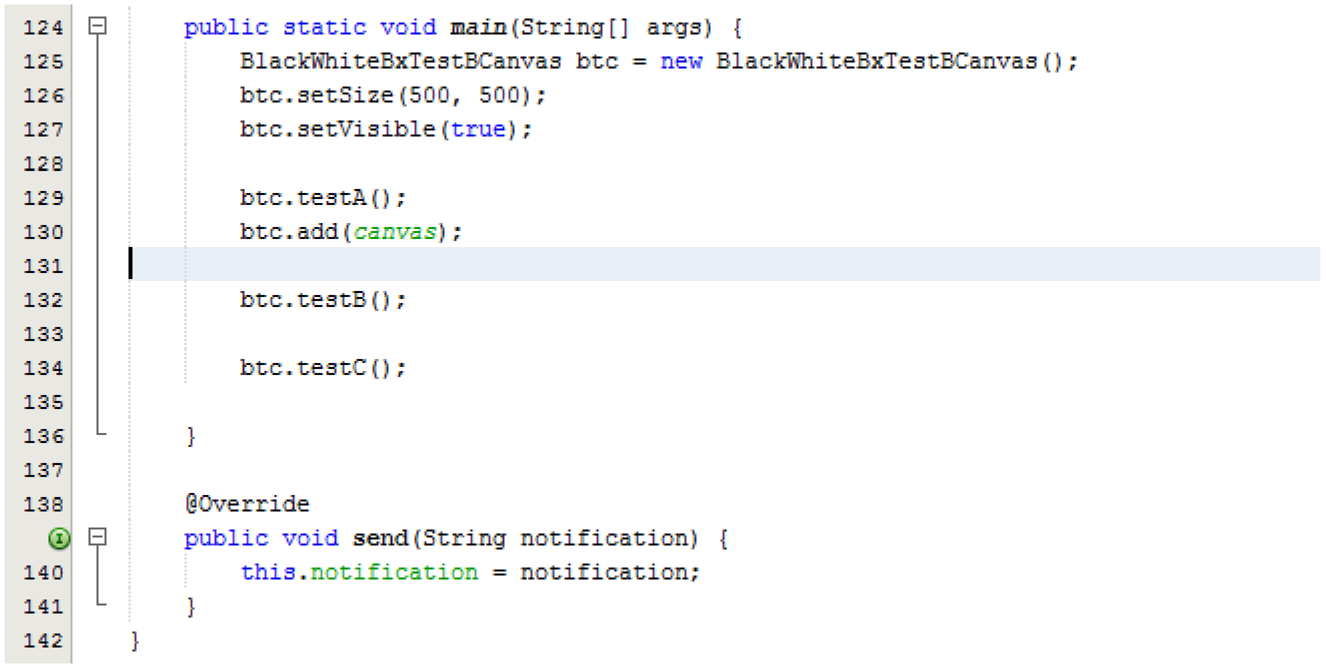
<u>Source Code</u>



```java
package test;

import cardstacks.Canvas;
import cardstacks.Card;
import cardstacks.CardStack;
import cardstacks.CardStackDealtCards;
import cardstacks.CardStackRemovedCards;
import cardstacks.CollectionCardStacks;
import cardstacks.Dice;
import cardstacks.DrawActionListener;
import cardstacks.NotationReader;
import cardstacks.NotificationListener;
import java.awt.Graphics;
import java.util.ArrayList;
import javax.swing.JFrame;

/**
 *
 * @author Biju Ale
 */
public class BlackWhiteBxTestBCanvas extends JFrame implements NotificationListener {

    static Canvas canvas;
    String notification;
    Graphics g;
    static DrawActionListener drawActionListener;

    void testA() {//For Test case #1
        canvas = new Canvas();
        drawActionListener = canvas;
        canvas.addNotificationListener(this);

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;
```

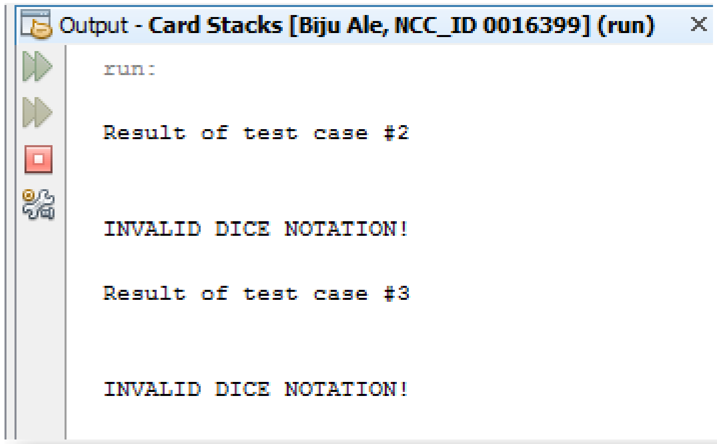


```java

        try {
            nreader.parseDiceNotation("4d4");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);

            //Simulating multiple card dealing
            ArrayList<Card> allDealtCards = new ArrayList();
            for (int i = 0; i < 4; i++) {
                allDealtCards.add(ccs.moveDealtCard(nreader.getDiceNotation(), csdc));
            }
            canvas.sendAllDealtCards(allDealtCards);
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_DEALT_CARD;
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    void testB() {//For Test case #2
        System.out.println("\nResult of test case #2");
        canvas = new Canvas();
        drawActionListener = canvas;
        canvas.addNotificationListener(this);

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("xxx");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);

            //Simulating multiple card dealing
            ArrayList<Card> allDealtCards = new ArrayList();
            for (int i = 0; i < 4; i++) {
                allDealtCards.add(ccs.moveDealtCard(nreader.getDiceNotation(), csdc));
            }
            canvas.sendAllDealtCards(allDealtCards);
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_DEALT_CARD;

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    void testC() {//For Test case #3
        System.out.println("\nResult of test case #3");
        canvas = new Canvas();
        drawActionListener = canvas;
        canvas.addNotificationListener(this);

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);

            //Simulating multiple card dealing
            ArrayList<Card> allDealtCards = new ArrayList();
            for (int i = 0; i < 4; i++) {
                allDealtCards.add(ccs.moveDealtCard(nreader.getDiceNotation(), csdc));
            }
            canvas.sendAllDealtCards(allDealtCards);
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_DEALT_CARD;

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

```



```
    public static void main(String[] args) {
        BlackWhiteBxTestBCanvas btc = new BlackWhiteBxTestBCanvas();
        btc.setSize(500, 500);
        btc.setVisible(true);

        btc.testA();
        btc.add(canvas);

        btc.testB();

        btc.testC();

    }

    @Override
    public void send(String notification) {
        this.notification = notification;
    }
}
```

Output



```
run:

Result of test case #2


INVALID DICE NOTATION!

Result of test case #3


INVALID DICE NOTATION!
```

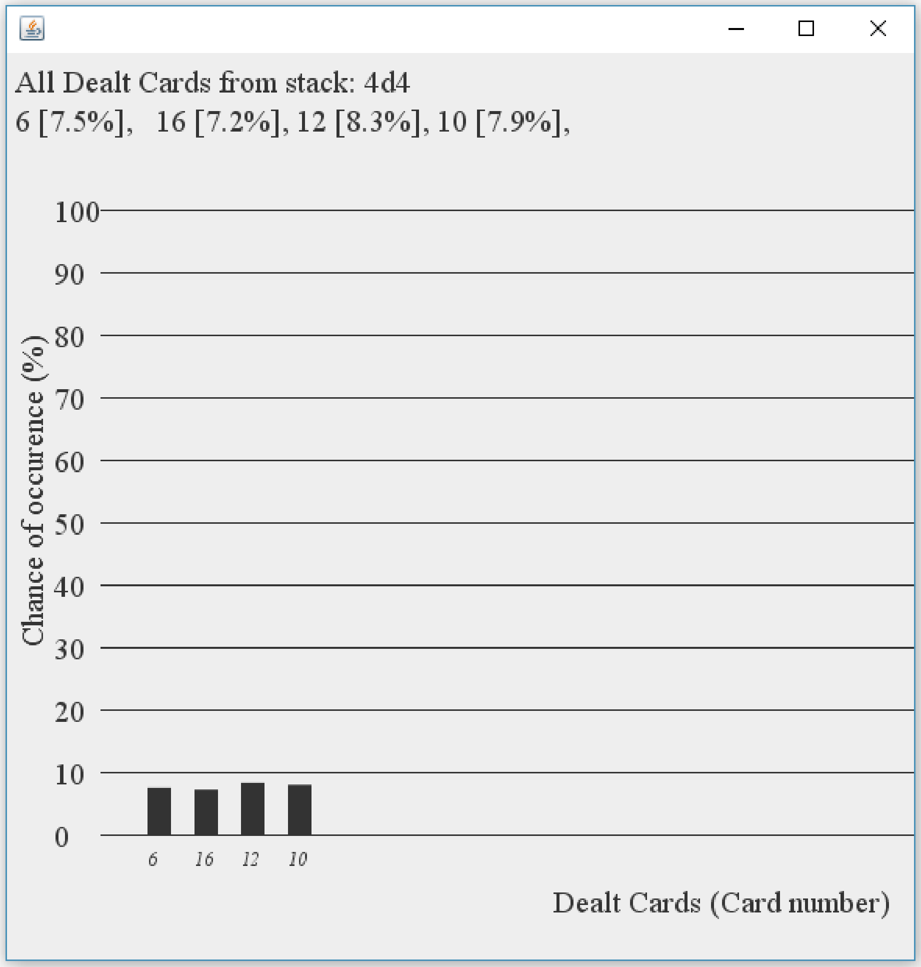

All Dealt Cards from stack: 4d4
6 [7.5%],   16 [7.2%], 12 [8.3%], 10 [7.9%],

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.

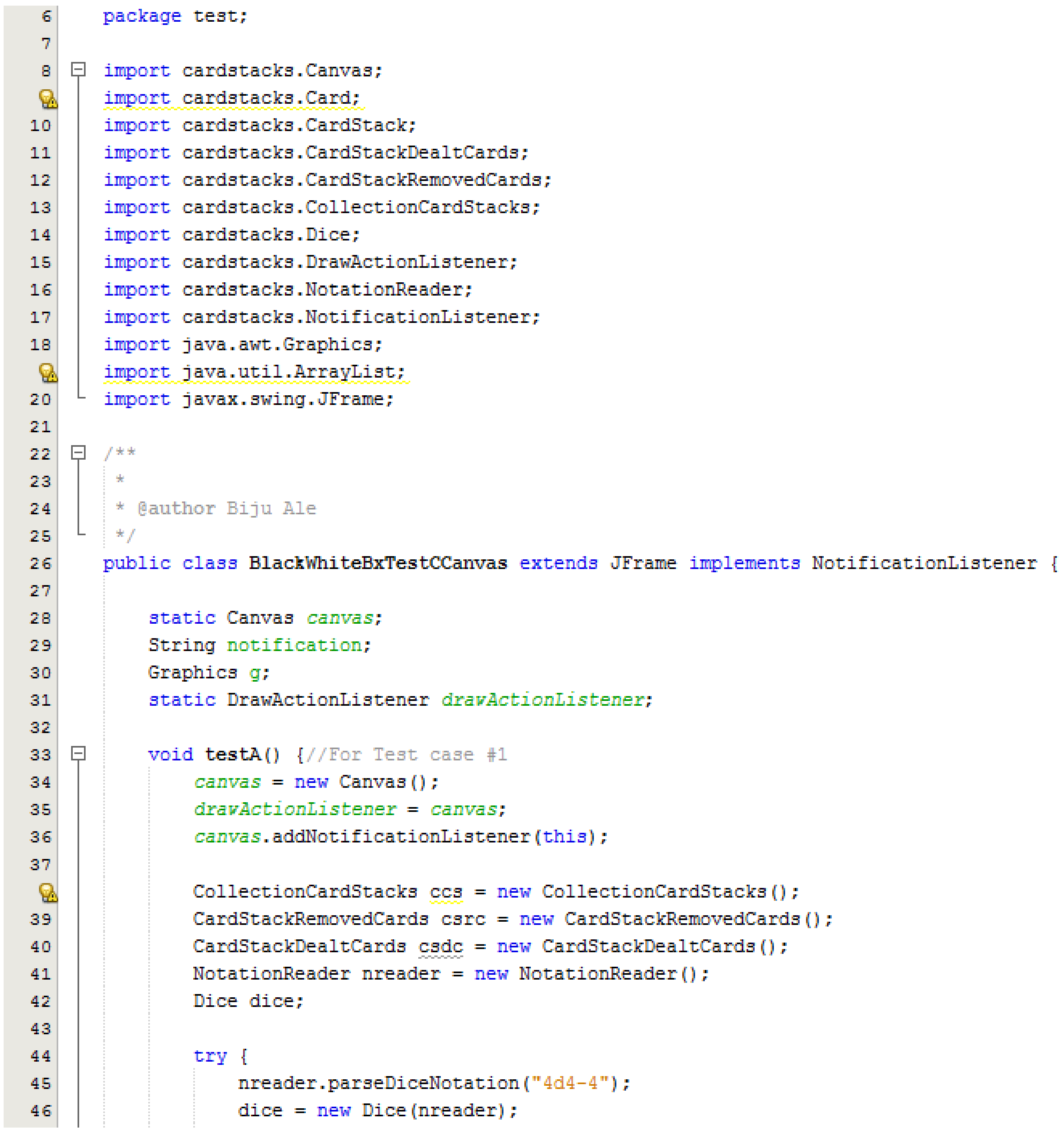
Test Suite No.23

Testingclass: cardstacks.Canvas
Testingtype: Black Box & White Box / Unit Testing

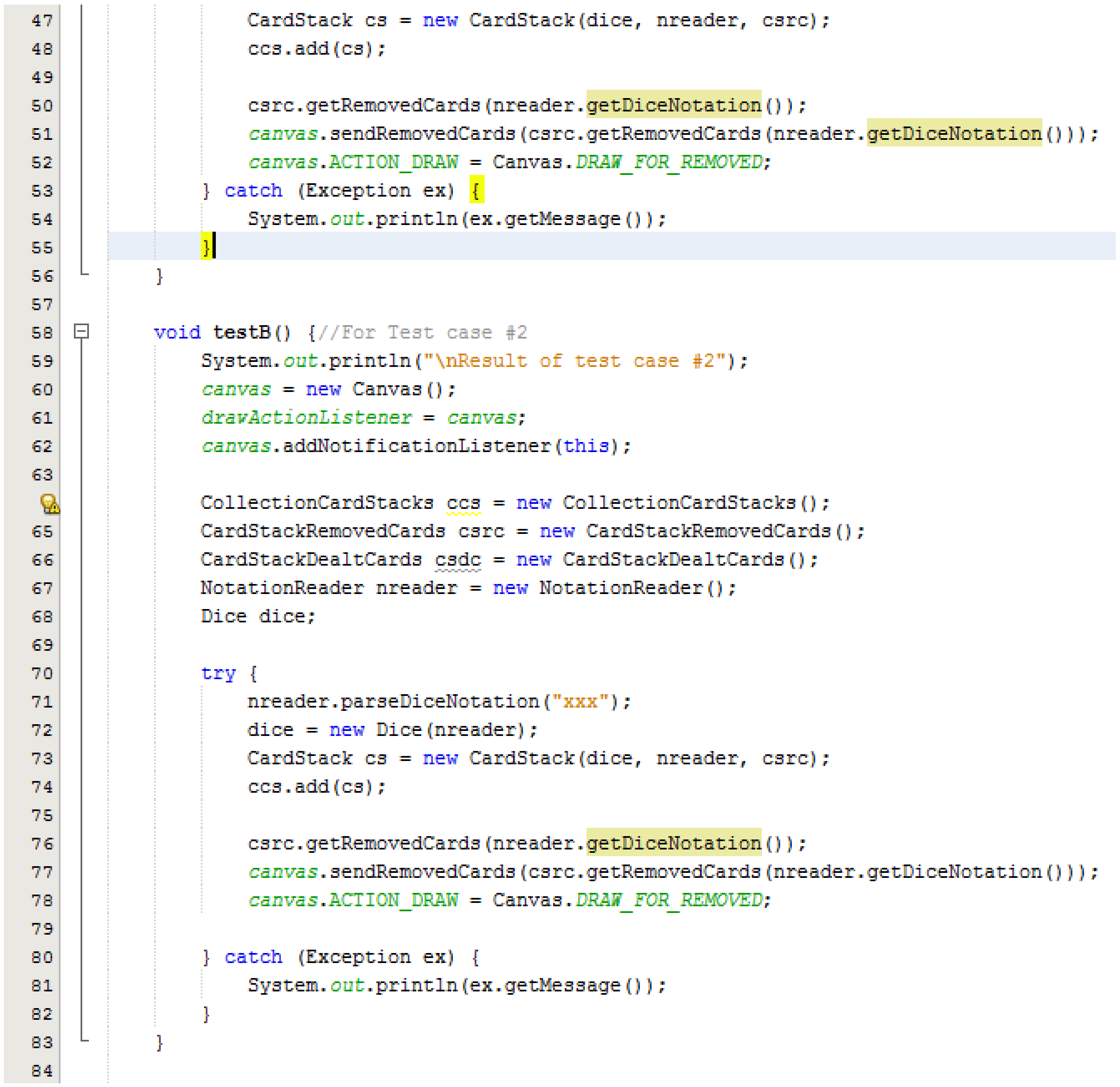
#	Method	Test Data	Input type	Expected Outcome
1	sendRemovedCards(ArrayList allRemovedCards)	allRemovedCards	Valid object	Draws all removed cards on JPanel
2	sendRemovedCards(ArrayList allRemovedCards)	allRemovedCards	Null (object has null data member i.e. null dice notation)	Throws exception with message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

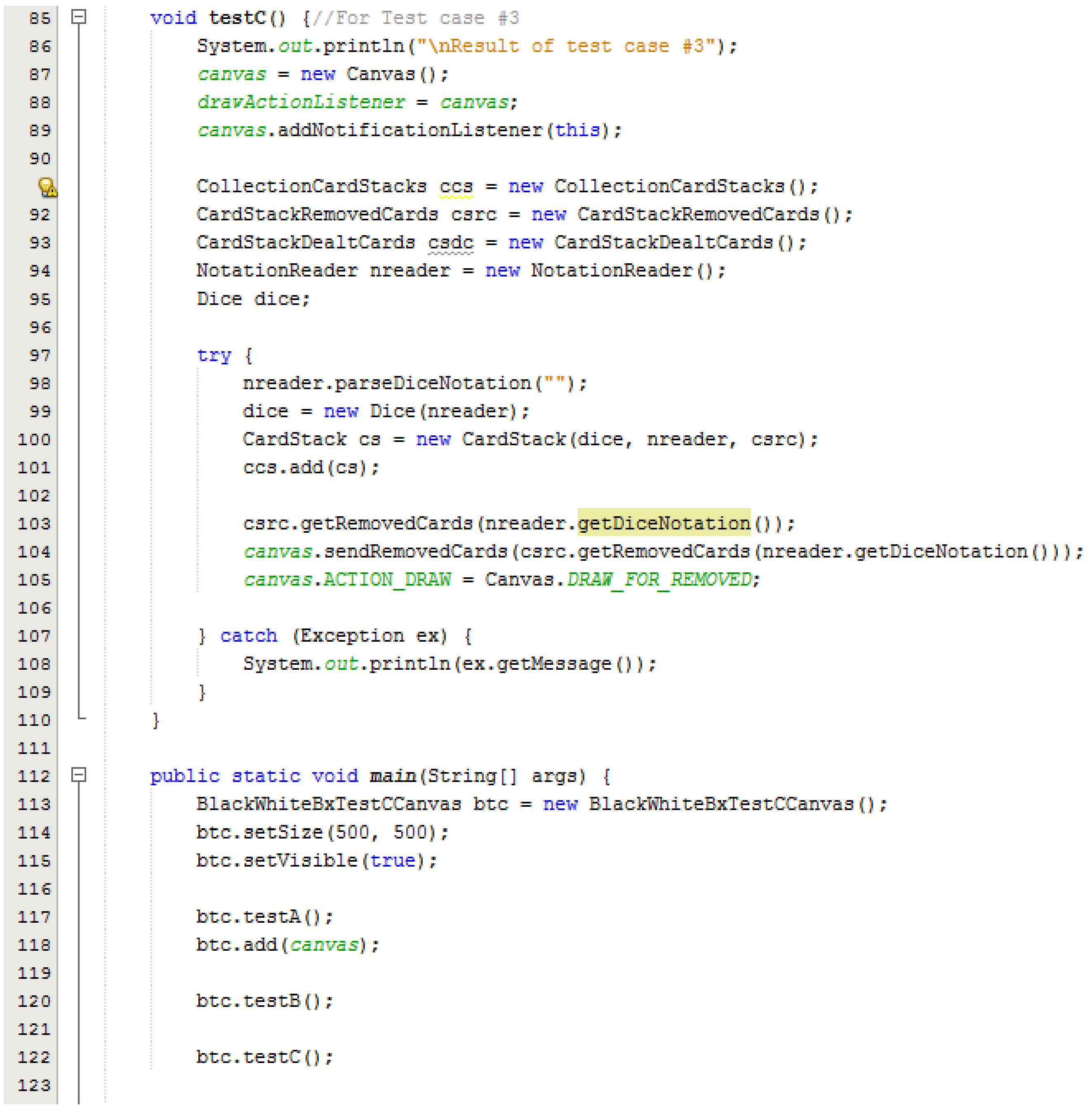
<u>Source Code</u>



```java
package test;

import cardstacks.Canvas;
import cardstacks.Card;
import cardstacks.CardStack;
import cardstacks.CardStackDealtCards;
import cardstacks.CardStackRemovedCards;
import cardstacks.CollectionCardStacks;
import cardstacks.Dice;
import cardstacks.DrawActionListener;
import cardstacks.NotationReader;
import cardstacks.NotificationListener;
import java.awt.Graphics;
import java.util.ArrayList;
import javax.swing.JFrame;

/**
 *
 * @author Biju Ale
 */
public class BlackWhiteBxTestCCanvas extends JFrame implements NotificationListener {

    static Canvas canvas;
    String notification;
    Graphics g;
    static DrawActionListener drawActionListener;

    void testA() {//For Test case #1
        canvas = new Canvas();
        drawActionListener = canvas;
        canvas.addNotificationListener(this);

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("4d4-4");
            dice = new Dice(nreader);
```

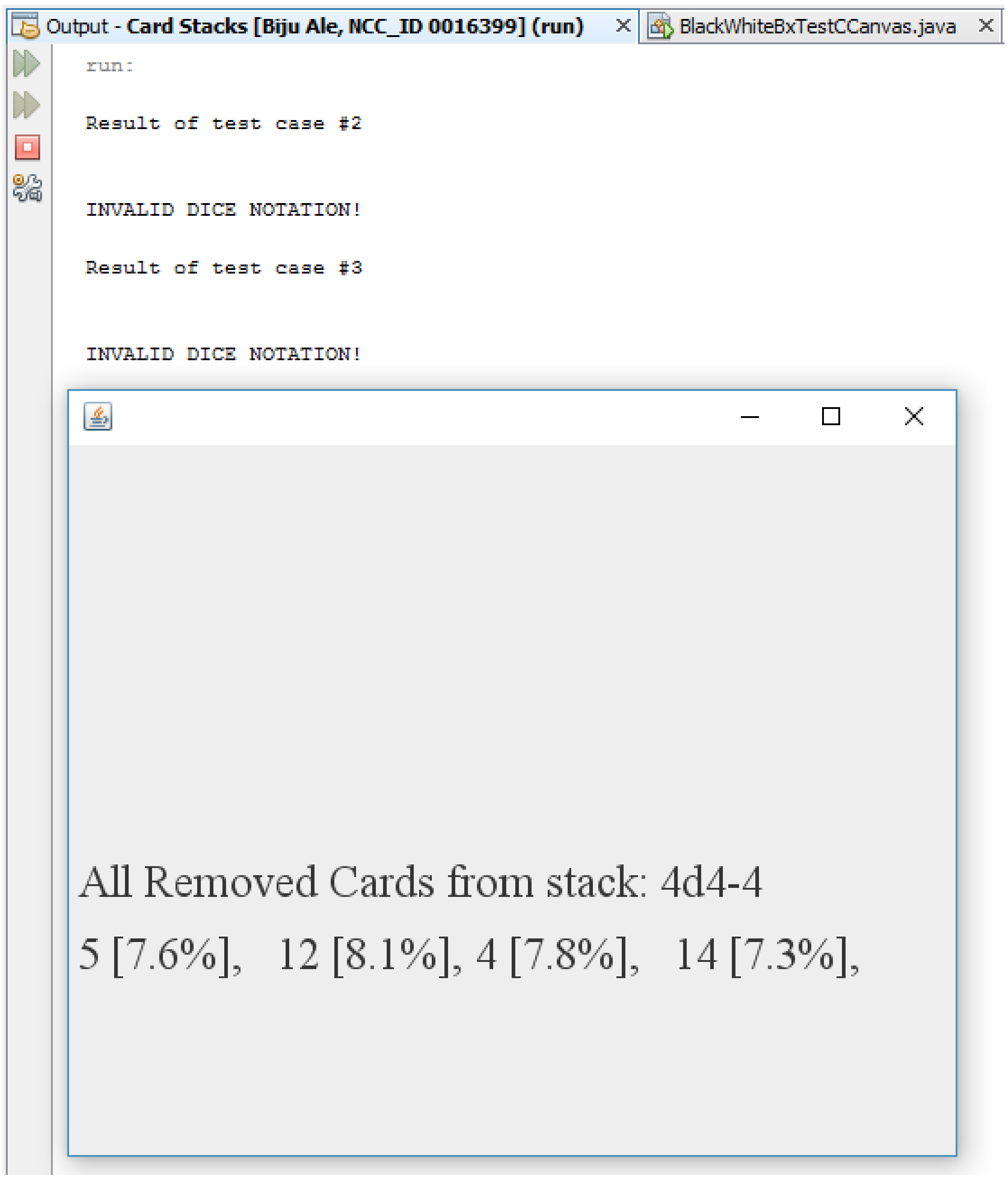


```
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);

            csrc.getRemovedCards(nreader.getDiceNotation());
            canvas.sendRemovedCards(csrc.getRemovedCards(nreader.getDiceNotation()));
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_REMOVED;
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    void testB() {//For Test case #2
        System.out.println("\nResult of test case #2");
        canvas = new Canvas();
        drawActionListener = canvas;
        canvas.addNotificationListener(this);

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("xxx");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);

            csrc.getRemovedCards(nreader.getDiceNotation());
            canvas.sendRemovedCards(csrc.getRemovedCards(nreader.getDiceNotation()));
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_REMOVED;

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
```



```java
    void testC() {//For Test case #3
        System.out.println("\nResult of test case #3");
        canvas = new Canvas();
        drawActionListener = canvas;
        canvas.addNotificationListener(this);

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);

            csrc.getRemovedCards(nreader.getDiceNotation());
            canvas.sendRemovedCards(csrc.getRemovedCards(nreader.getDiceNotation()));
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_REMOVED;

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    public static void main(String[] args) {
        BlackWhiteBxTestCCanvas btc = new BlackWhiteBxTestCCanvas();
        btc.setSize(500, 500);
        btc.setVisible(true);

        btc.testA();
        btc.add(canvas);

        btc.testB();

        btc.testC();

```

Output


Output - Card Stacks [Biju Ale, NCC_ID 0016399] (run)
BlackWhiteBxTestCCanvas.java

```
run:

Result of test case #2


INVALID DICE NOTATION!

Result of test case #3


INVALID DICE NOTATION!
```

All Removed Cards from stack: 4d4-4

5 [7.6%],   12 [8.1%], 4 [7.8%],   14 [7.3%],

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.

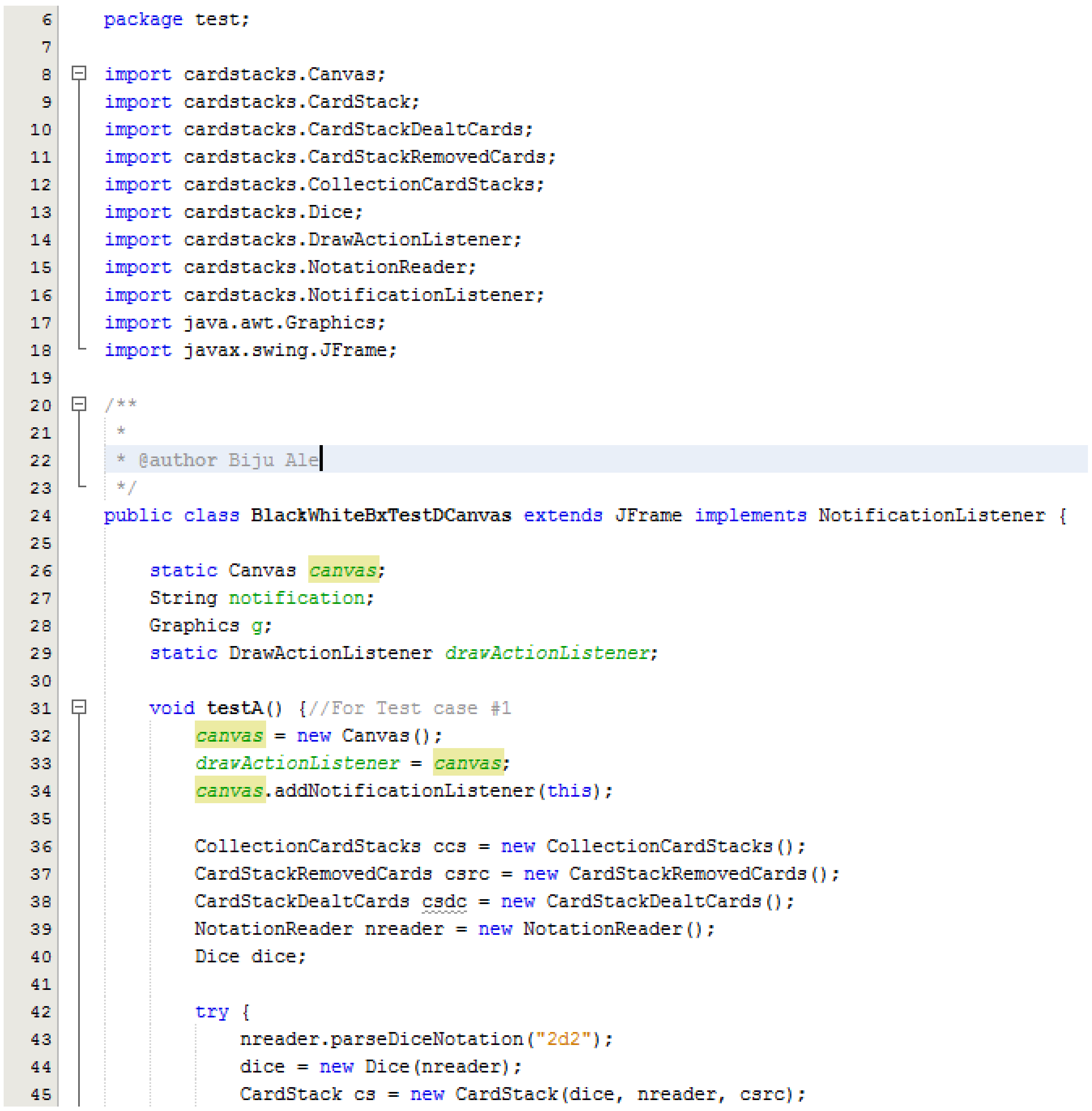
Test Suite No.24

Testingclass: cardstacks.Canvas
Testingtype: Black Box & White Box / Unit Testing

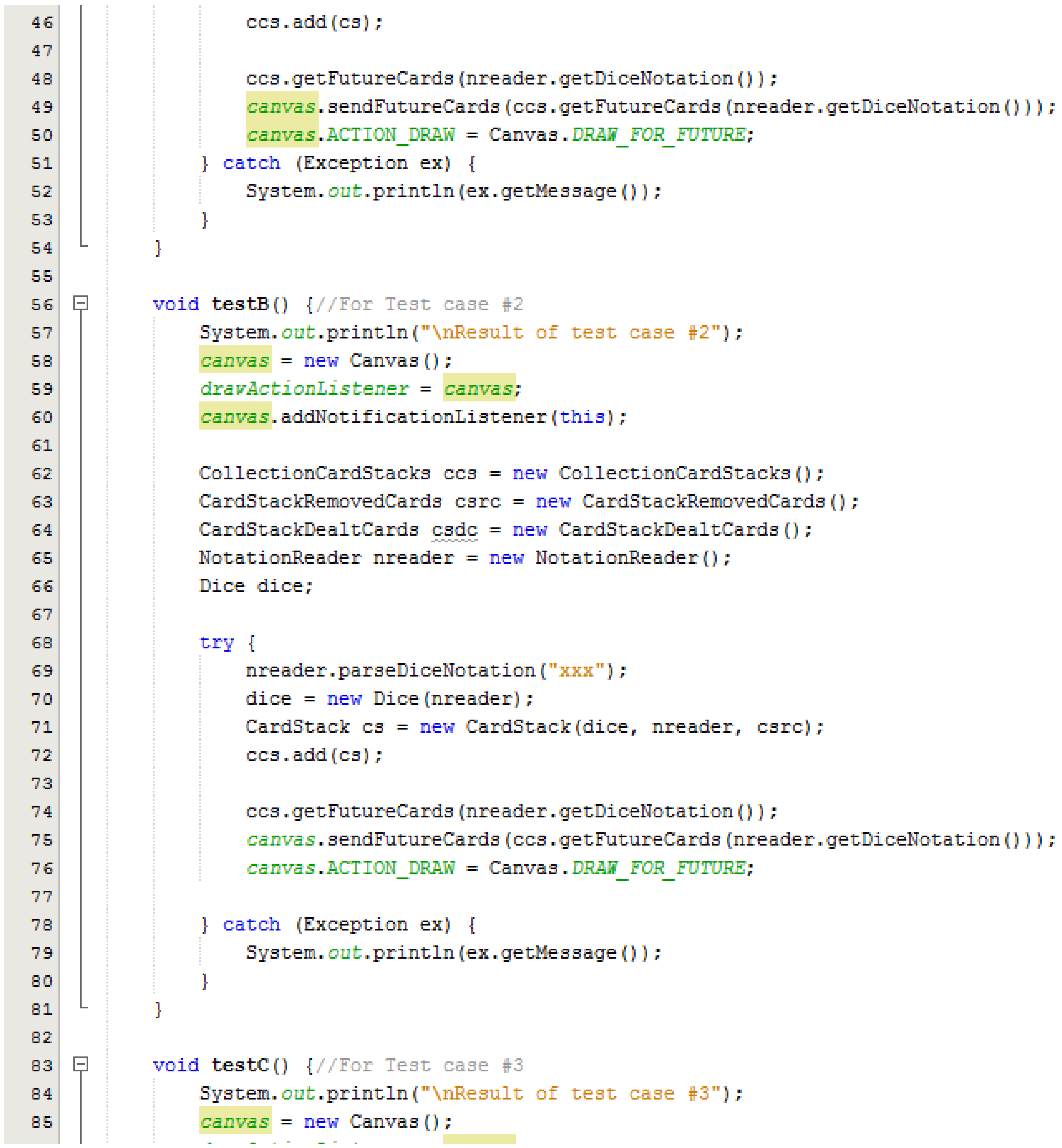
#	Method	Test Data	Input type	Expected Outcome
1	sendFutureCards(CardStack futureCards)	futureCards	Valid object	Draws graph of all future cards on JPanel
2	sendFutureCards(CardStack futureCards)	futureCards	Null (object has null data member i.e. null dice notation)	Throws exception with message – “INVALID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

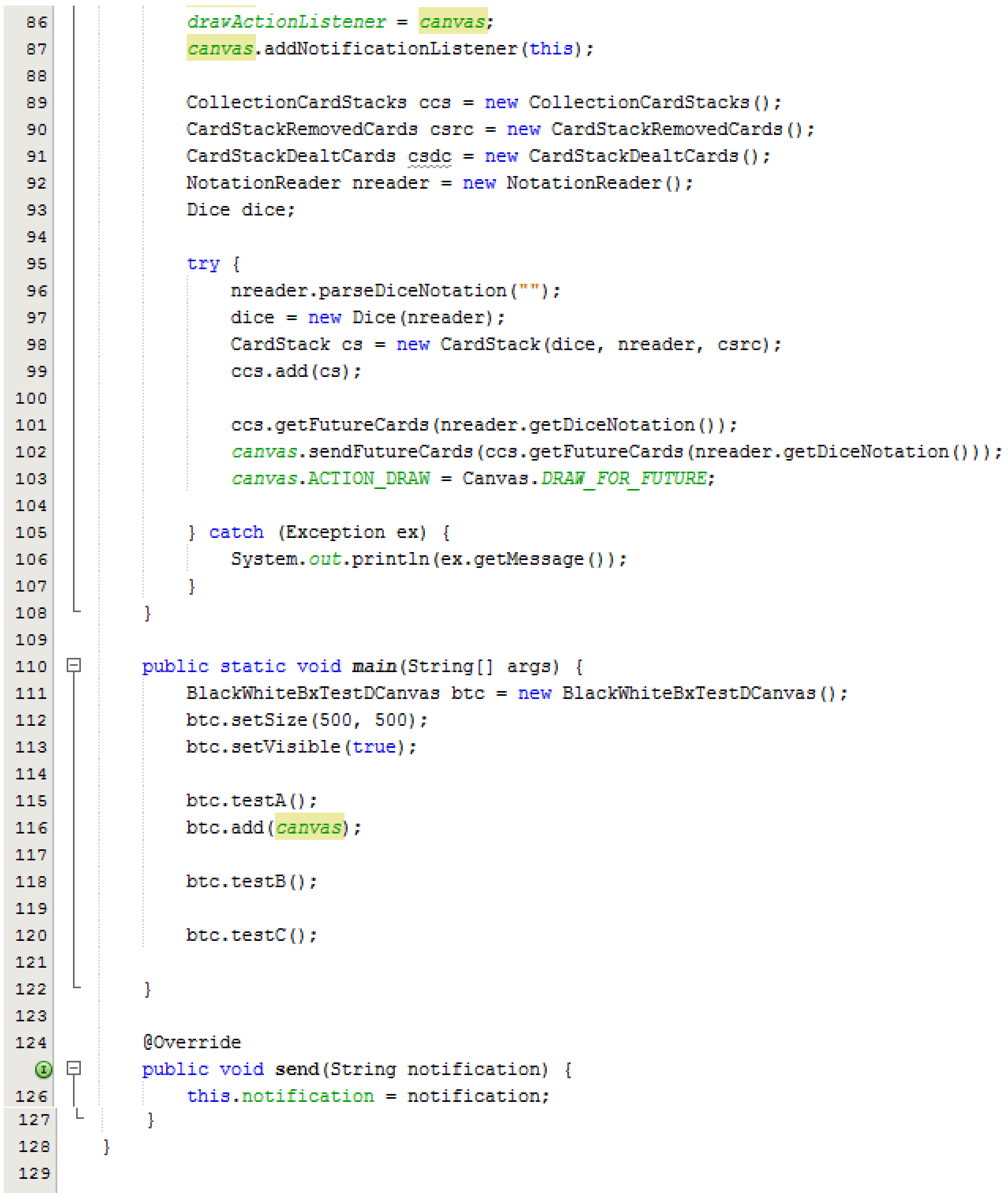
<u>Source Code</u>



```java
package test;

import cardstacks.Canvas;
import cardstacks.CardStack;
import cardstacks.CardStackDealtCards;
import cardstacks.CardStackRemovedCards;
import cardstacks.CollectionCardStacks;
import cardstacks.Dice;
import cardstacks.DrawActionListener;
import cardstacks.NotationReader;
import cardstacks.NotificationListener;
import java.awt.Graphics;
import javax.swing.JFrame;

/**
 *
 * @author Biju Ale
 */
public class BlackWhiteBxTestDCanvas extends JFrame implements NotificationListener {

    static Canvas canvas;
    String notification;
    Graphics g;
    static DrawActionListener drawActionListener;

    void testA() {//For Test case #1
        canvas = new Canvas();
        drawActionListener = canvas;
        canvas.addNotificationListener(this);

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("2d2");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
```

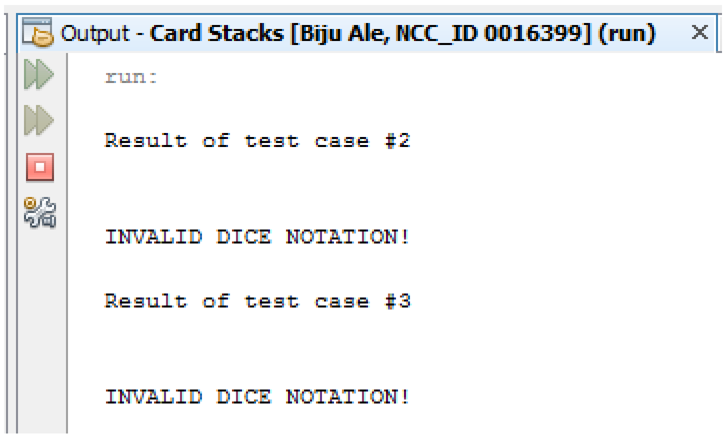


```
            ccs.add(cs);

            ccs.getFutureCards(nreader.getDiceNotation());
            canvas.sendFutureCards(ccs.getFutureCards(nreader.getDiceNotation()));
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_FUTURE;
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    void testB() {//For Test case #2
        System.out.println("\nResult of test case #2");
        canvas = new Canvas();
        drawActionListener = canvas;
        canvas.addNotificationListener(this);

        CollectionCardStacks ccs = new CollectionCardStacks();
        CardStackRemovedCards csrc = new CardStackRemovedCards();
        CardStackDealtCards csdc = new CardStackDealtCards();
        NotationReader nreader = new NotationReader();
        Dice dice;

        try {
            nreader.parseDiceNotation("xxx");
            dice = new Dice(nreader);
            CardStack cs = new CardStack(dice, nreader, csrc);
            ccs.add(cs);

            ccs.getFutureCards(nreader.getDiceNotation());
            canvas.sendFutureCards(ccs.getFutureCards(nreader.getDiceNotation()));
            canvas.ACTION_DRAW = Canvas.DRAW_FOR_FUTURE;

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    void testC() {//For Test case #3
        System.out.println("\nResult of test case #3");
        canvas = new Canvas();
```



```
            drawActionListener = canvas;
            canvas.addNotificationListener(this);

            CollectionCardStacks ccs = new CollectionCardStacks();
            CardStackRemovedCards csrc = new CardStackRemovedCards();
            CardStackDealtCards csdc = new CardStackDealtCards();
            NotationReader nreader = new NotationReader();
            Dice dice;

            try {
                nreader.parseDiceNotation("");
                dice = new Dice(nreader);
                CardStack cs = new CardStack(dice, nreader, csrc);
                ccs.add(cs);

                ccs.getFutureCards(nreader.getDiceNotation());
                canvas.sendFutureCards(ccs.getFutureCards(nreader.getDiceNotation()));
                canvas.ACTION_DRAW = Canvas.DRAW_FOR_FUTURE;

            } catch (Exception ex) {
                System.out.println(ex.getMessage());
            }
        }

        public static void main(String[] args) {
            BlackWhiteBxTestDCanvas btc = new BlackWhiteBxTestDCanvas();
            btc.setSize(500, 500);
            btc.setVisible(true);

            btc.testA();
            btc.add(canvas);

            btc.testB();

            btc.testC();

        }

        @Override
        public void send(String notification) {
            this.notification = notification;
        }
    }
```

Output


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

```
run:

Result of test case #2


INVALID DICE NOTATION!

Result of test case #3


INVALID DICE NOTATION!
```

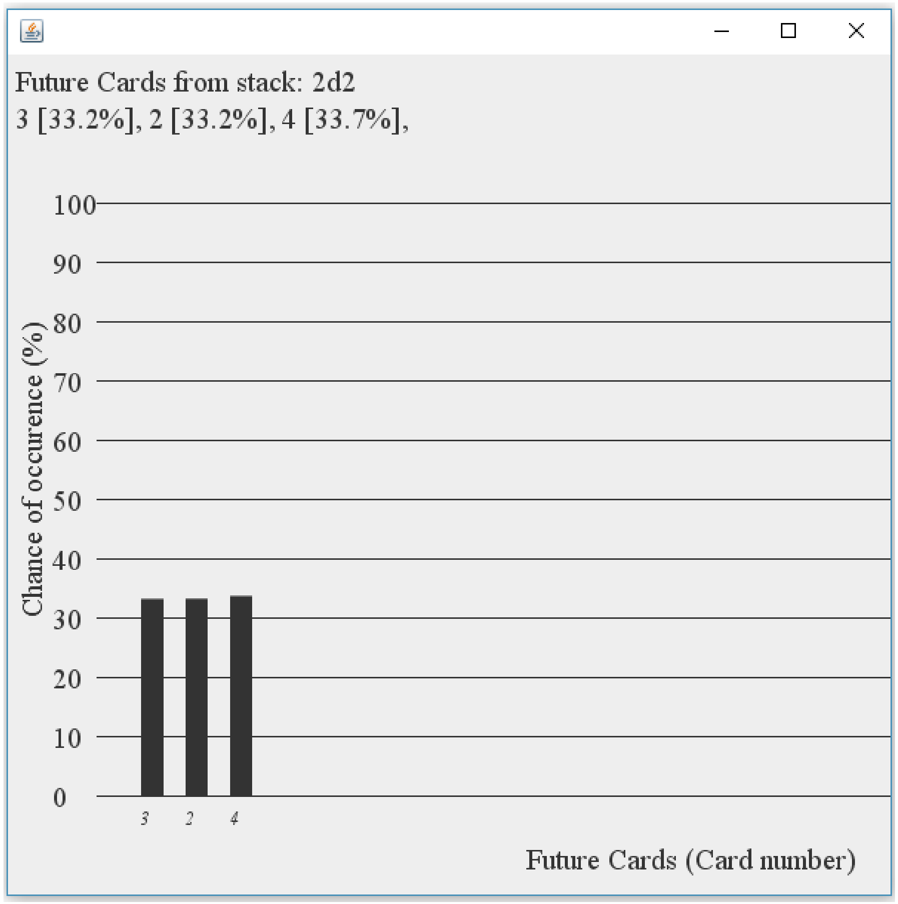

Future Cards from stack: 2d2
3 [33.2%], 2 [33.2%], 4 [33.7%],

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.

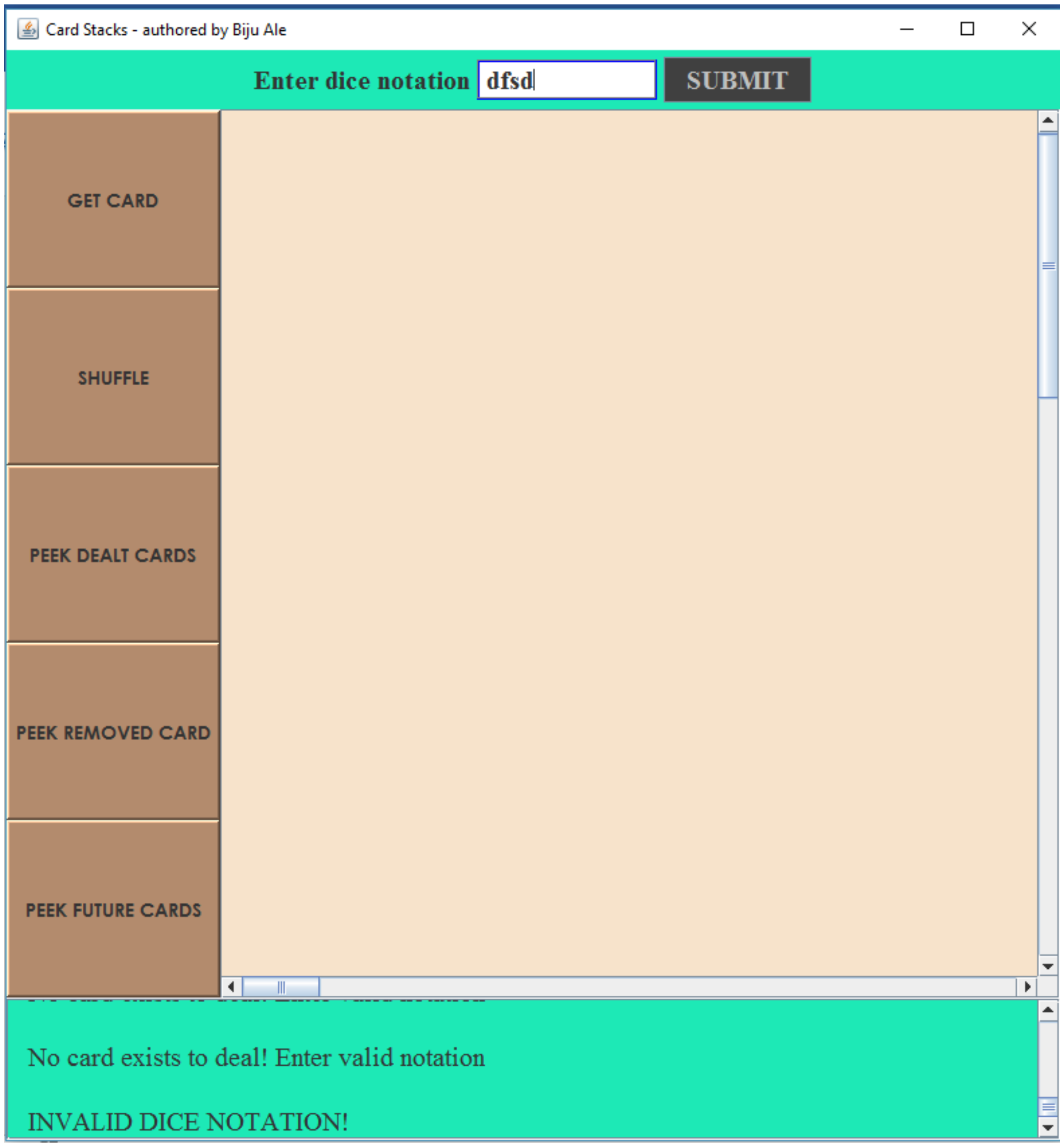
Final Test Suite No.25

Testingclass: cardstacks.NotationReader
Testingtype: Integration Testing

Test Suite No. 1 to 24 was designedtounit 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.

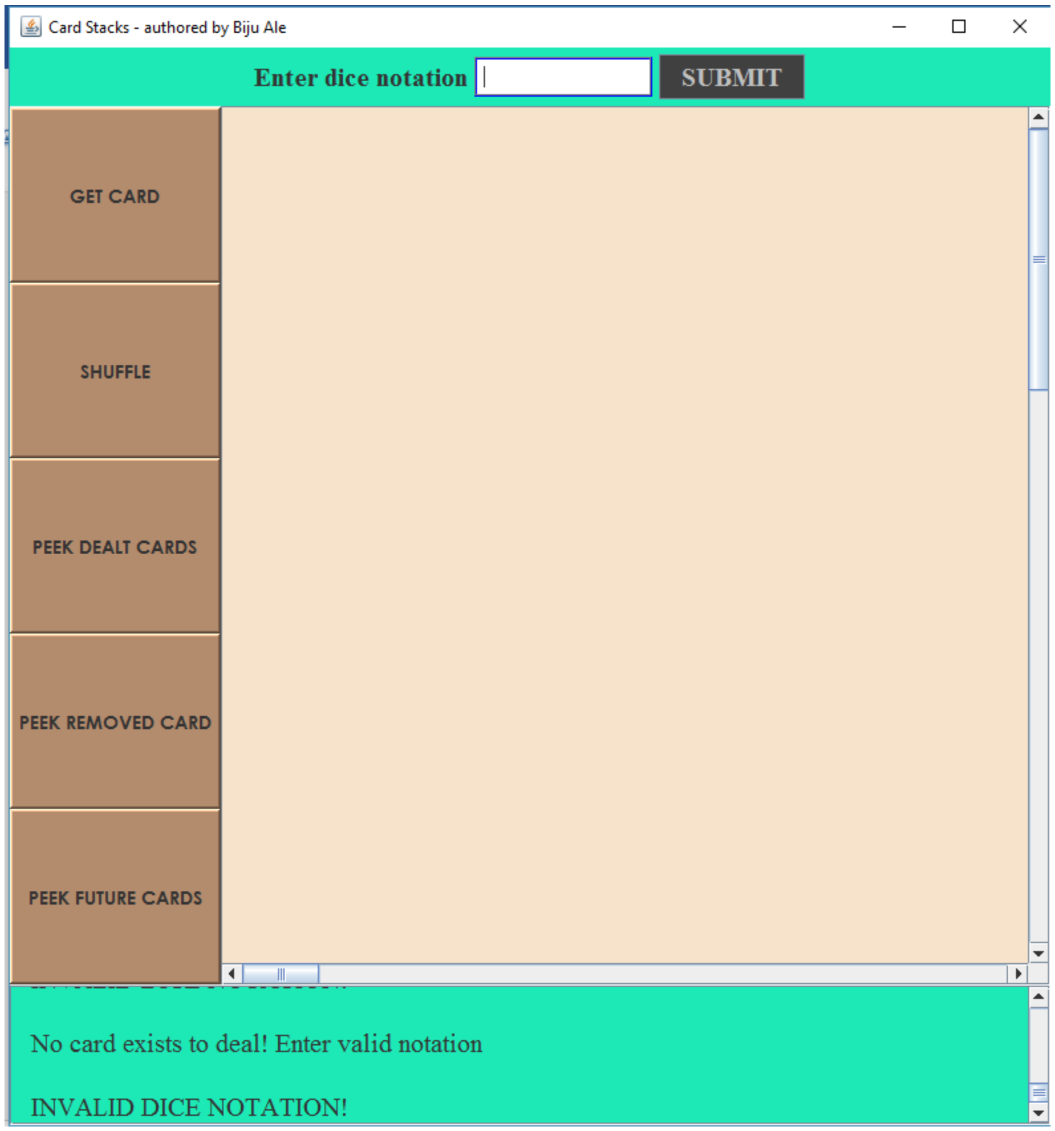
Test Execution

Please turn over….



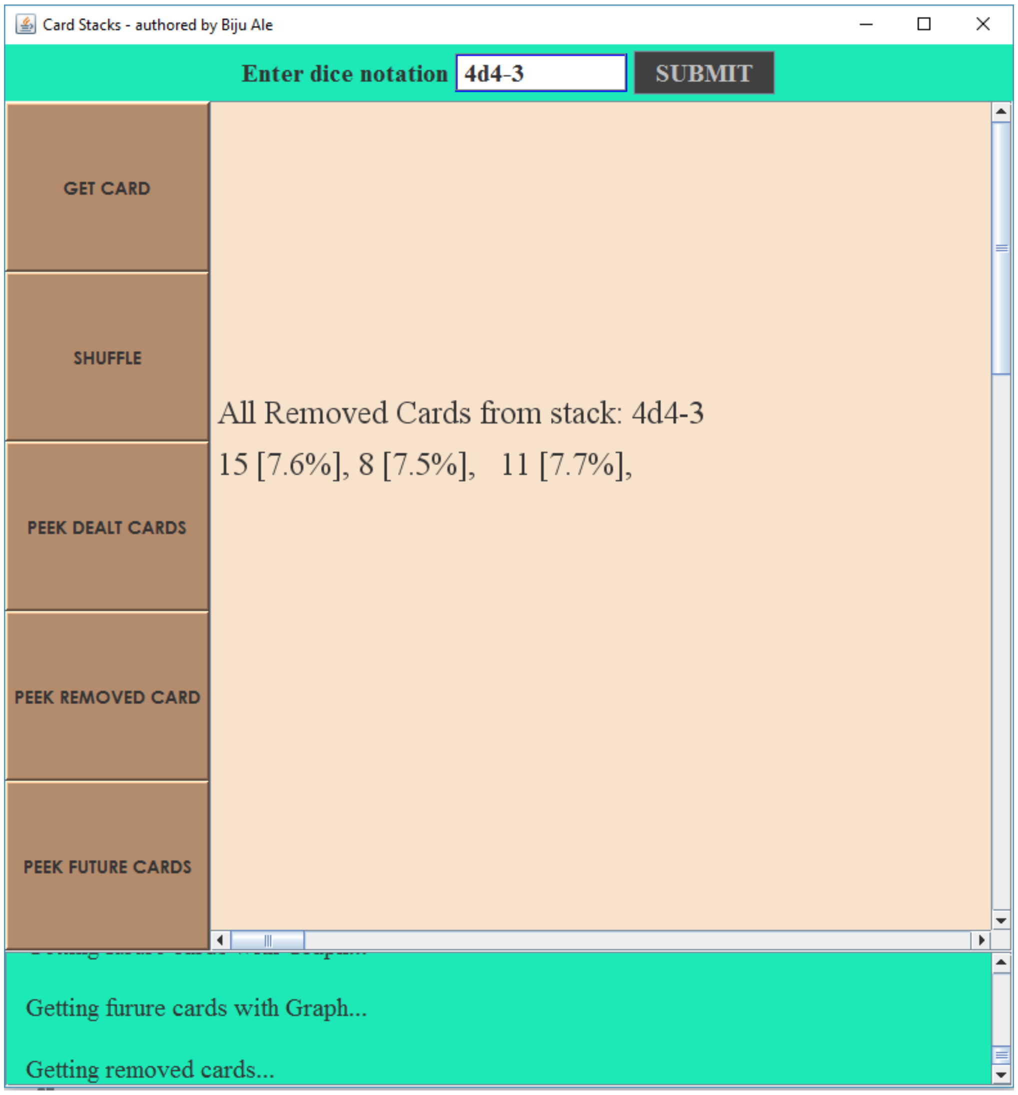
```
No card exists to deal! Enter valid notation

INVALID DICE NOTATION!
```

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



```
No card exists to deal! Enter valid notation

INVALID DICE NOTATION!
```

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

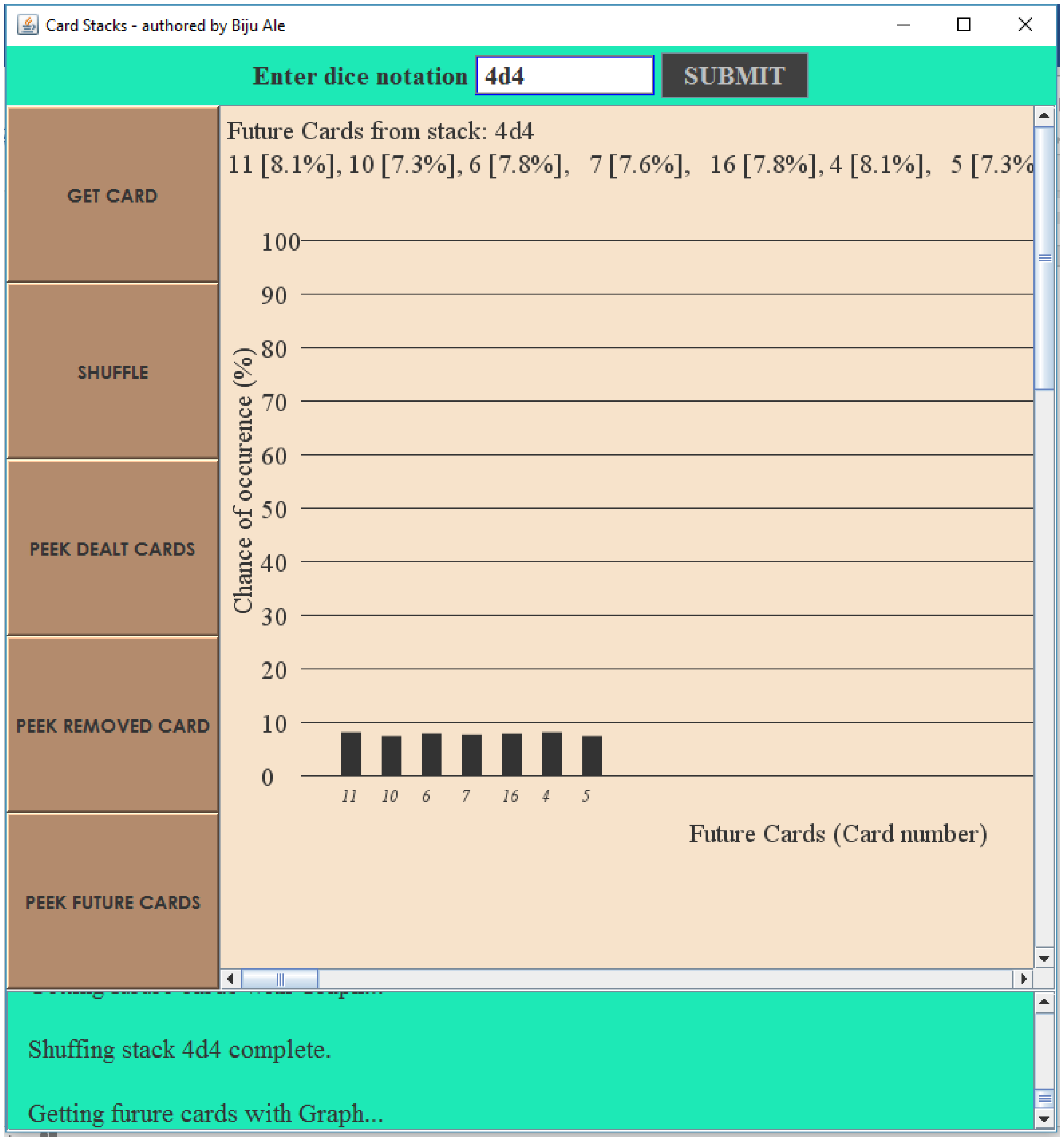


All Removed Cards from stack: 4d4-3
15 [7.6%], 8 [7.5%],   11 [7.7%],

Getting furure cards with Graph...

Getting removed cards...

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.

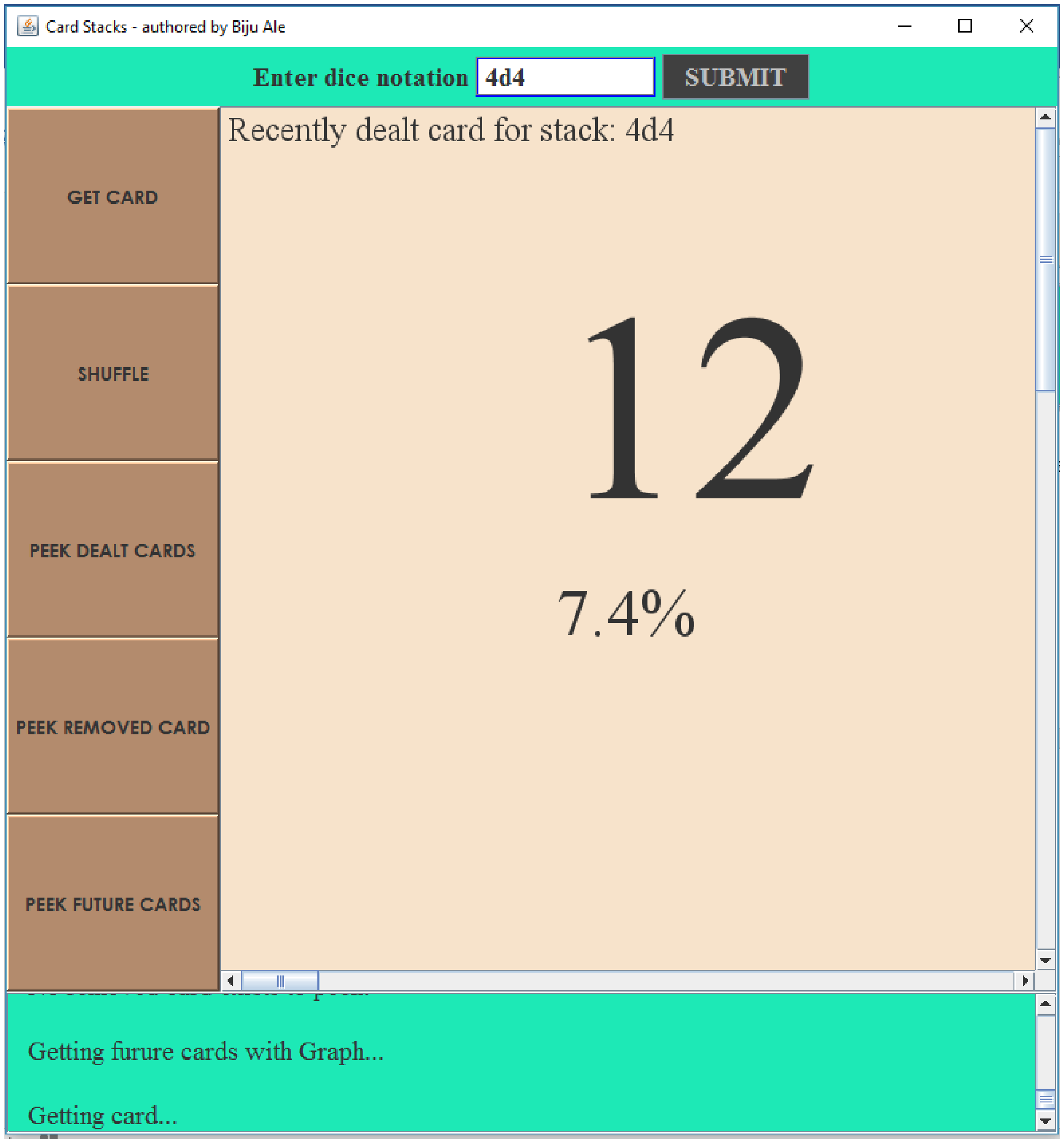


Future Cards from stack: 4d4
11 [8.1%], 10 [7.3%], 6 [7.8%], 7 [7.6%], 16 [7.8%], 4 [8.1%], 5 [7.3%

Shuffing stack 4d4 complete.

Getting furure cards with Graph...

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



Recently dealt card for stack: 4d4

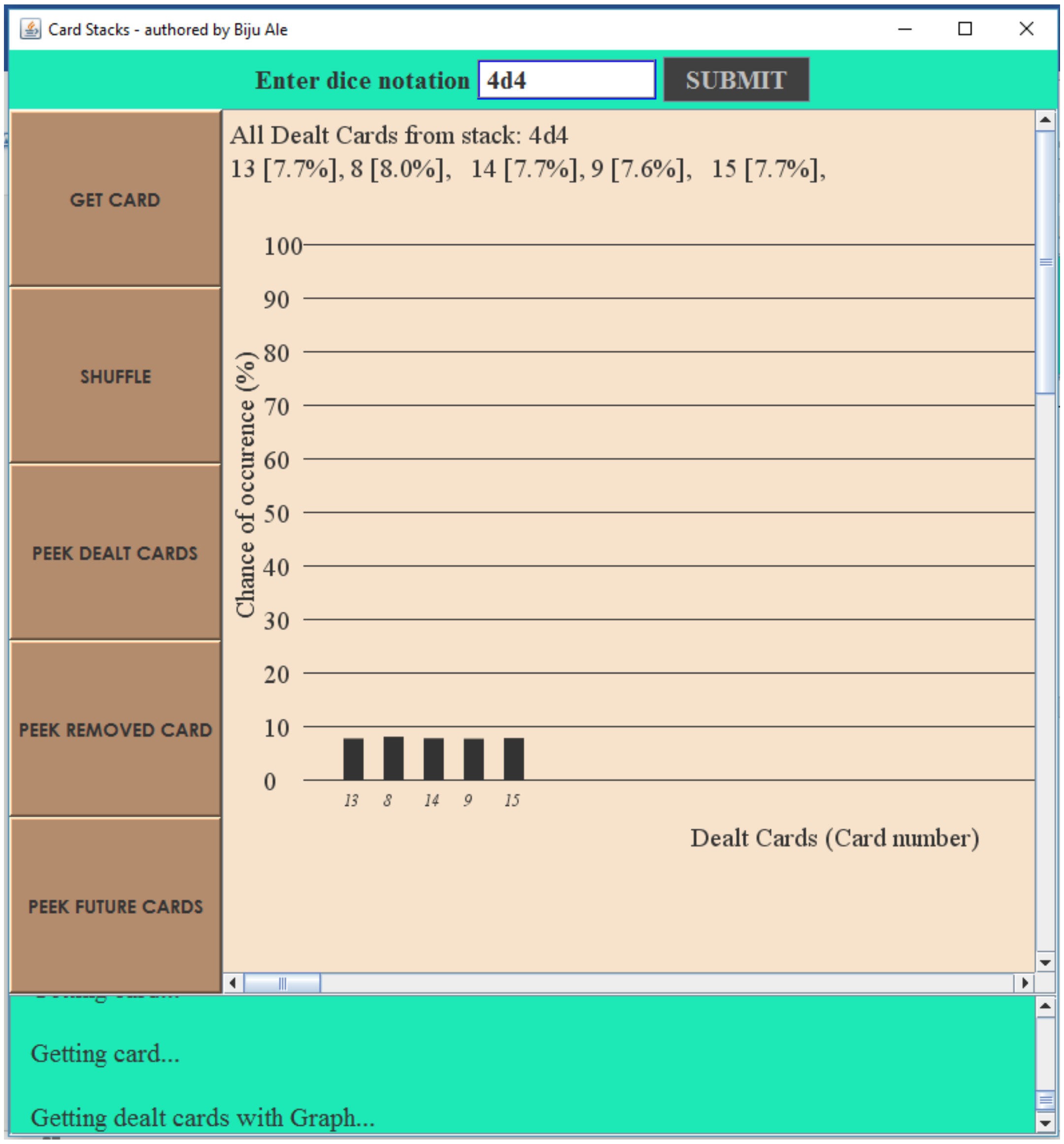
12

7.4%

Getting furure cards with Graph...

Getting card...

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



All Dealt Cards from stack: 4d4
13 [7.7%], 8 [8.0%], 14 [7.7%], 9 [7.6%], 15 [7.7%],

Getting card...

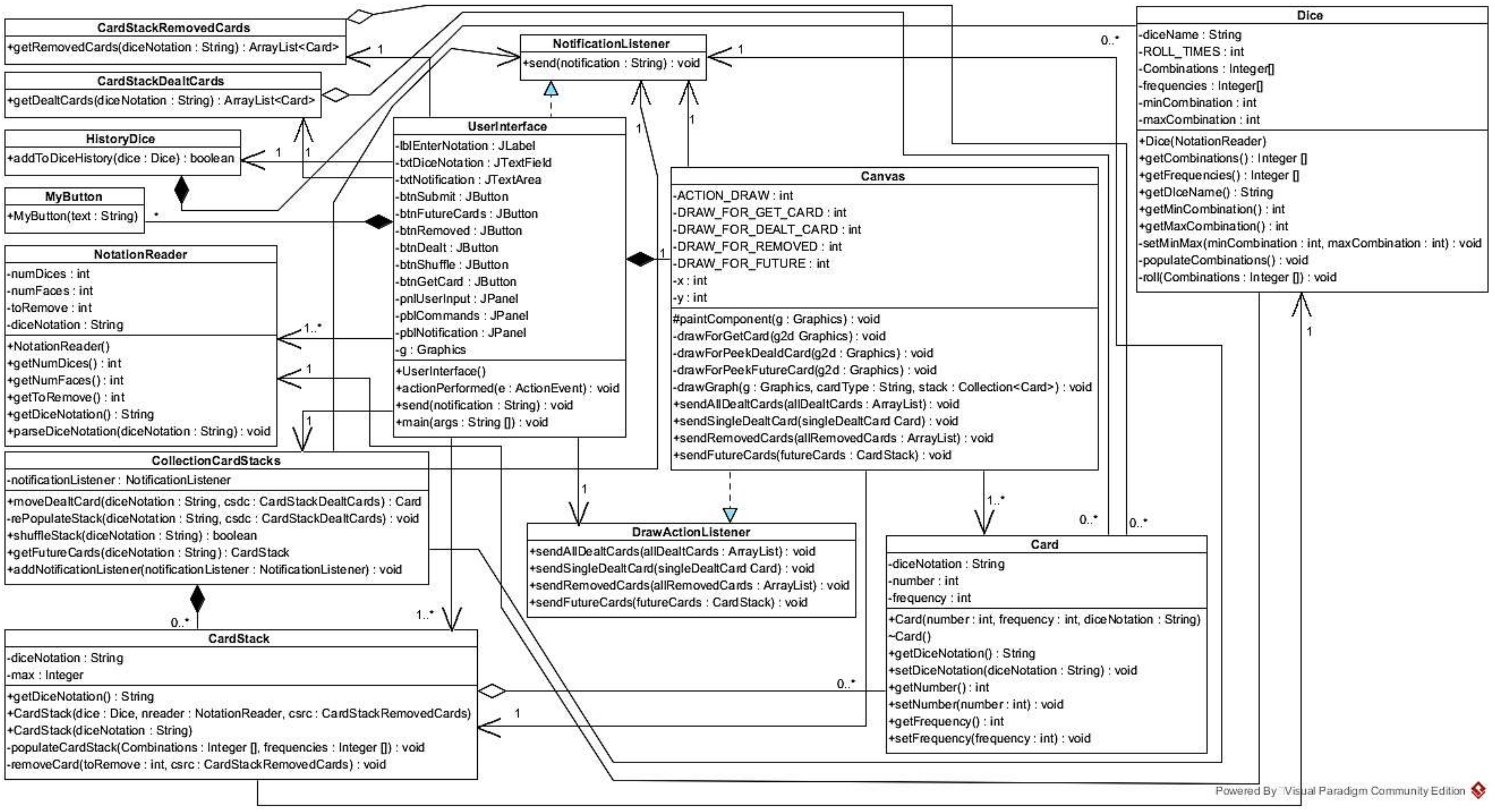
Getting dealt cards with Graph...

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.

Task 3 – Class Diagram


CardStackRemovedCards
+getRemovedCards(diceNotation : String) : ArrayList<Card>
CardStackDealtCards
+getDealtCards(diceNotation : String) : ArrayList<Card>
HistoryDice
+addToDiceHistory(dice : Dice) : boolean
MyButton
+MyButton(text : String)
NotationReader
-numDices : int
-numFaces : int
-toRemove : int
-diceNotation : String
+NotationReader()
+getNumDices() : int
+getNumFaces() : int
+getToRemove() : int
+getDiceNotation() : String
+parseDiceNotation(diceNotation : String) : void
NotificationListener
+send(notification : String) : void
UserInterface
-lblEnterNotation : JLabel
-txtDiceNotation : JTextField
-txtNotification : JTextArea
-btnSubmit : JButton
-btnFutureCards : JButton
-btnRemoved : JButton
-btnDealt : JButton
-btnShuffle : JButton
-btnGetCard : JButton
-pnlUserInput : JPanel
-pblCommands : JPanel
-pblNotification : JPanel
-g : Graphics
+UserInterface()
+actionPerformed(e : ActionEvent) : void
+send(notification : String) : void
+main(args : String []) : void
Canvas
-ACTION_DRAW : int
-DRAW_FOR_GET_CARD : int
-DRAW_FOR_DEALT_CARD : int
-DRAW_FOR_REMOVED : int
-DRAW_FOR_FUTURE : int
-x : int
-y : int
#paintComponent(g : Graphics) : void
-drawForGetCard(g2d Graphics) : void
-drawForPeekDealdCard(g2d : Graphics) : void
-drawForPeekFutureCard(g2d : Graphics) : void
-drawGraph(g : Graphics, cardType : String, stack : Collection<Card>) : void
+sendAllDealtCards(allDealtCards : ArrayList) : void
+sendSingleDealtCard(singleDealtCard Card) : void
+sendRemovedCards(allRemovedCards : ArrayList) : void
+sendFutureCards(futureCards : CardStack) : void
Dice
-diceName : String
-ROLL_TIMES : int
-Combinations : Integer[]
-frequencies : Integer[]
-minCombination : int
-maxCombination : int
+Dice(NotationReader)
+getCombinations() : Integer []
+getFrequencies() : Integer []
+getDIceName() : String
+getMinCombination() : int
+getMaxCombination() : int
-setMinMax(minCombination : int, maxCombination : int) : void
-populateCombinations() : void
-roll(Combinations : Integer []) : void
CollectionCardStacks
-notificationListener : NotificationListener
+moveDealtCard(diceNotation : String, csdc : CardStackDealtCards) : Card
-rePopulateStack(diceNotation : String, csdc : CardStackDealtCards) : void
+shuffleStack(diceNotation : String) : boolean
+getFutureCards(diceNotation : String) : CardStack
+addNotificationListener(notificationListener : NotificationListener) : void
DrawActionListener
+sendAllDealtCards(allDealtCards : ArrayList) : void
+sendSingleDealtCard(singleDealtCard Card) : void
+sendRemovedCards(allRemovedCards : ArrayList) : void
+sendFutureCards(futureCards : CardStack) : void
Card
-diceNotation : String
-number : int
-frequency : int
+Card(number : int, frequency : int, diceNotation : String)
~Card()
+getDiceNotation() : String
+setDiceNotation(diceNotation : String) : void
+getNumber() : int
+setNumber(number : int) : void
+getFrequency() : int
+setFrequency(frequency : int) : void
CardStack
-diceNotation : String
-max : Integer
+getDiceNotation() : String
+CardStack(dice : Dice, nreader : NotationReader, csrc : CardStackRemovedCards)
+CardStack(diceNotation : String)
-populateCardStack(Combinations : Integer [], frequencies : Integer []) : void
-removeCard(toRemove : int, csrc : CardStackRemovedCards) : void
1
0..*
1..*
Powered By Visual Paradigm Community Edition

Page **136** of **140**

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.