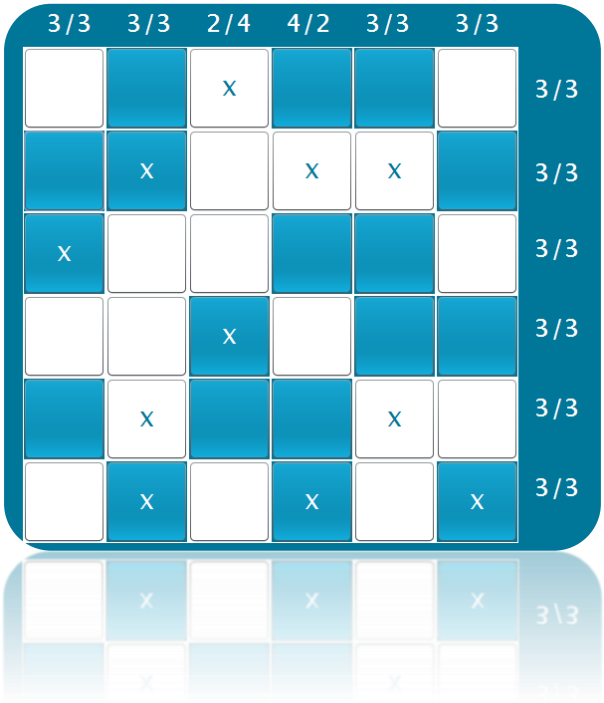
3-IN-A-ROW

Project 1



JASMINE ANICA

Java Programming: Data Structures

42030

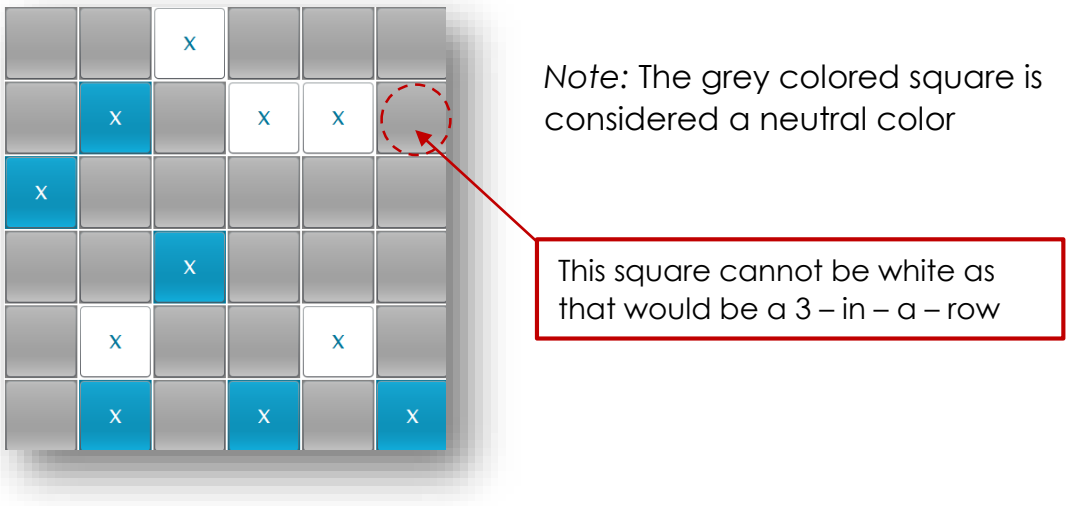
INTRODUCTION

Title: 3 – in – a – row

Originated in Japan, this simple puzzle game is based on pure logic. A board of 6 x 6 is placed with the colors Blue, Gray, and White squares. The goal of 3-in-a-row is to manipulate the board such that there are three Blue squares and three White squares in each row and column. Oddly enough, the title of the game refers to a rule that cannot be broken: there must not be a 3-in-a-row of the same color. I found myself enjoying this lesser known type of puzzle enough to program it myself.

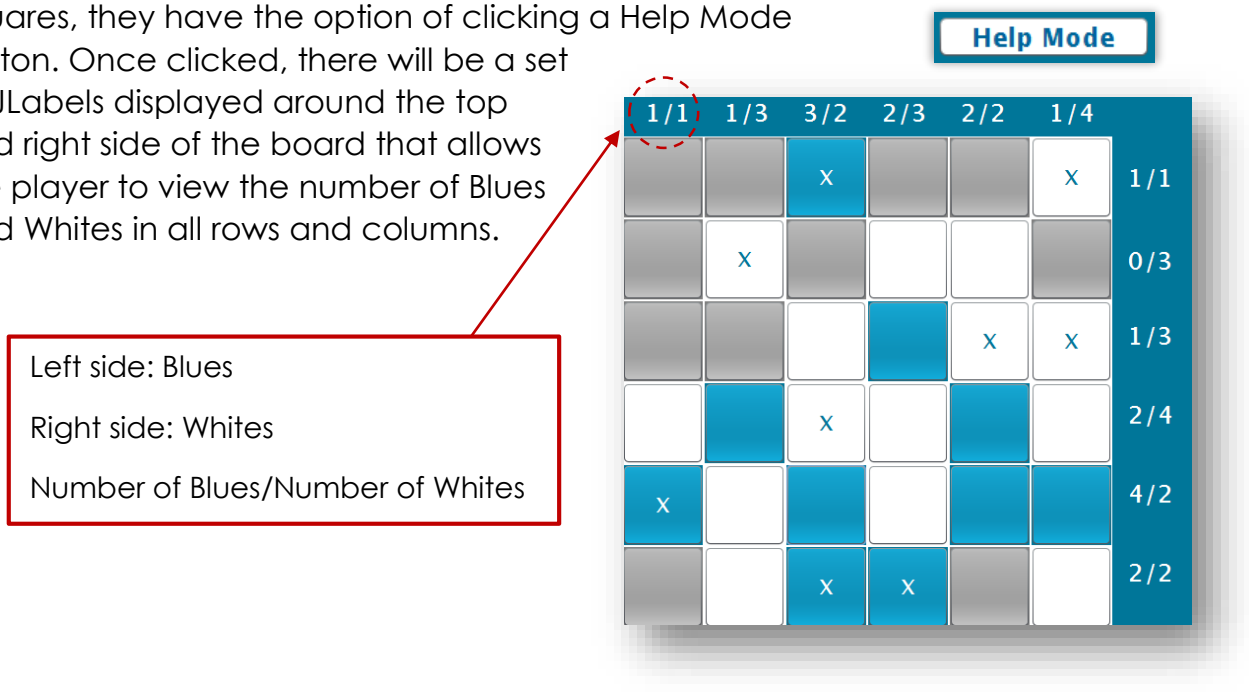
GAMEPLAY/RULES

As stated above, the main objective is to change the squares on the board so the number of blue and white squares are equal in both rows and columns. The player can simply click on a square to change the color in this consecutive order: G->B->W. The squares with a mark 'X' are fixed (cannot be changed) to the board so the player must work around those squares. The most **important rule** of the game is there ***cannot*** be a 3–in–a–row of the same color (this includes the marked squares).



GAMEPLAY/RULES (CONT.)

If the Player does not want to keep counting the numbers of Blue and White squares, they have the option of clicking a Help Mode Button. Once clicked, there will be a set of JLabels displayed around the top and right side of the board that allows the player to view the number of Blues and Whites in all rows and columns.



SUMMARY

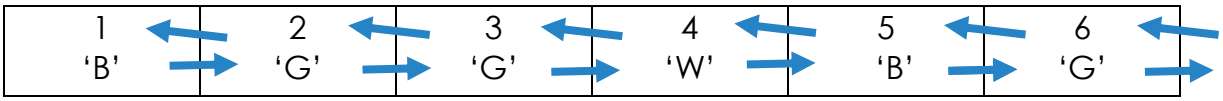
Lines of Code	**1926**
Lines of Comments	**501**
Number of Methods	**49**
Number of Classes	**6**

I created this Java program using NetBeans IDE 8.02 and utilized NetBeans Swing GUI builder. The part I focused on was holding the values of a board in a LinkedList. Replacing a 2D array, the DoublyLinkedList holds a number representing an index from 1- 36 and a character representing a color for the JButton.

DOUBLYLINKEDLIST TREATED AS A BOARD

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

I used this representation to count how many blues and whites in each row in increments of 6. In reality the list looks like this

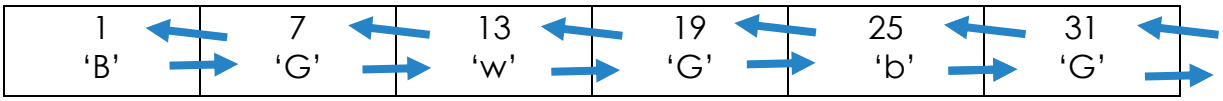


(Etc.)

However to calculate the number of blues and whites in each column I had to create another Linked List that represents the columns as another 2D array shown below

DOUBLYLINKEDLIST TREATED AS A BOARD IN ORDER OF COLUMNS

1	7	13	19	25	31
2	8	14	20	26	32
3	9	15	21	27	33
4	10	16	22	28	34
5	11	17	23	29	35
6	12	18	24	30	36



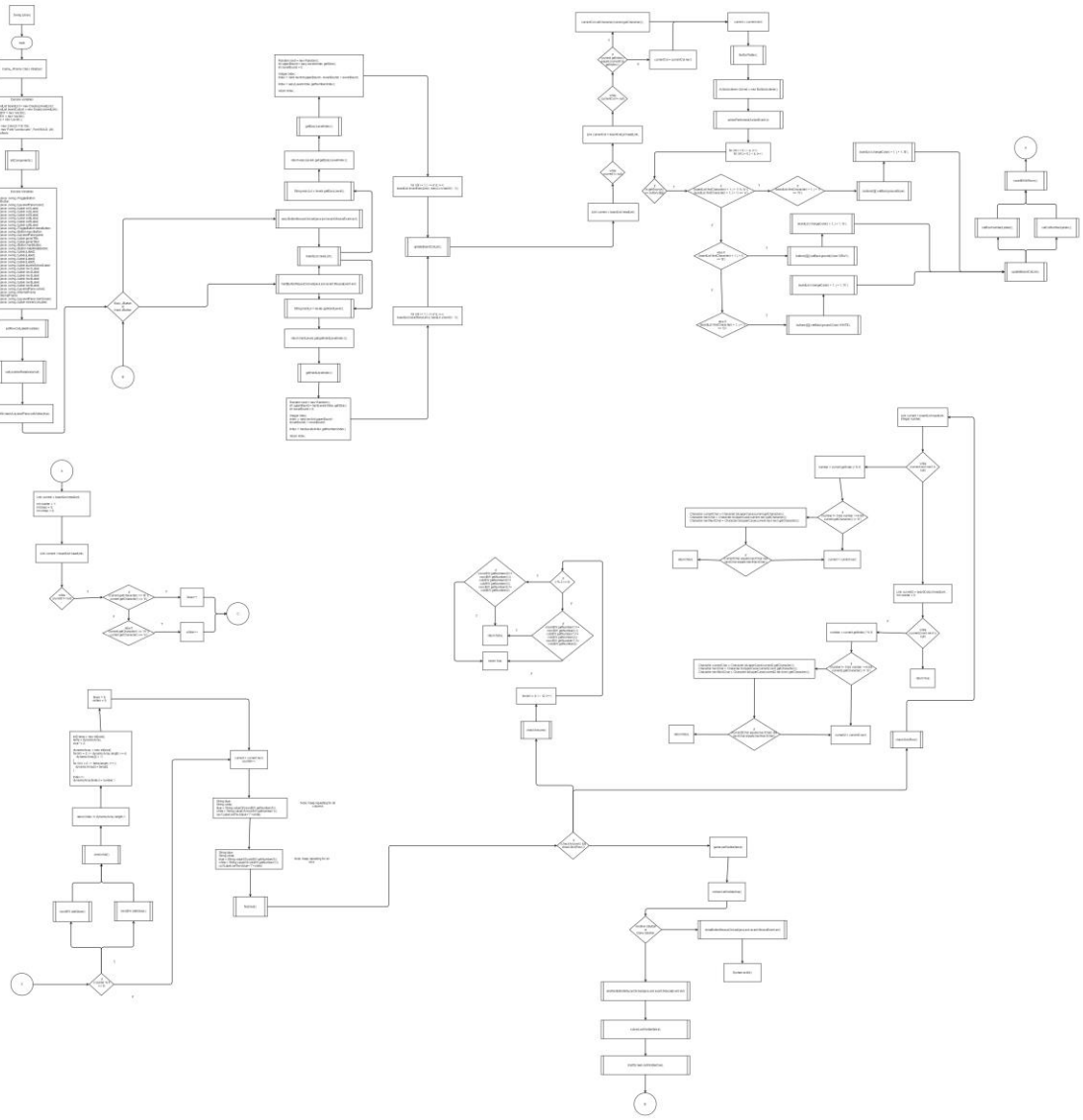
(Etc.)

Then I focused on counting the number of 'B' for blue and 'W' for white characters using both Linked Lists. I stored these values in a Vector to refer to later. My next focus was to enforce the rule that a solved board cannot have a three in a row of the same color. This proved to be quite easy because I was able to access the references to the Link next to the first Link, and the Link next

to the Link next to the first Link (a tongue twister) in order to catch any three in a rows with the same characters.

As an added bonus, I stored the different levels in a text file and added them to a HashMap. Each level consists of a 36 character long String that also represents the board as the Linked List have above. I created a hash function that created a unique key to store the String value in.

FLOW CHART



PSEUDO CODE

Initialize

If Player clicks on the Easy level JButton

The Easy_Level text file is read from line by line and saved into a HashMap

Board DoublyLinkedList saves the information for a level from the HashMap

A JButton 6 x 6 grid layout is created and added to a board JPanel

Based on the board DoublyLinkedList, method is called to fix squares

If the corresponding index is equal to the JButton index and board DoublyLinkedList contains a 'b'

Set JButton background to blue

Set JButton text to 'X'

If the corresponding index is equal to the JButton index and board DoublyLinkedList contains a 'w'

Set JButton background to white

Set JButton text to 'X'

JLabels will be set by counting the number of blue and white squares on the board

Game JLayeredPane will appear while start JLayeredPane will hide

If the player clicks any JButton on the board

ActionListener will check which button was clicked

If the corresponding index of the button is equal to the index of the board DoublyLinkedList and contains character 'G'

Change the button to blue

Set the character from 'G' to 'B'

If the corresponding index of the button is equal to the index of the board DoublyLinkedList and contains character 'B'

Change the button to white

Set the character from 'B' to 'W'

If the corresponding index of the button is equal to the index of the board DoublyLinkedList and contains character 'W'

PSEUDO CODE (CONT.)

Change the button to grey

Set the character from 'W' to 'G'

Update the change in color to the board DoublyLinkedList

Update the JLabels new values

If player finished the game

Game JLayeredPane hides, solved JLayeredPane appears

If the another button is clicked

The player is sent back to the beginning screen to choose an easy or hard level

If the done button is clicked

The program will exit

If the player clicks the help button

The JLabels surrounding the board becomes visible to show the number of blues and whites in each row and column

If the player clicks the help button again

The JLabels surrounding the board becomes invisible

If Player clicks on the Hard level JButton

The Hard_Level text file is read from line by line and saved into a HashMap

Board DoublyLinkedList saves the information for a level from the HashMap

A JButton 6 x 6 grid layout is created and added to a board JPanel

Based on the board DoublyLinkedList, method is called to fix squares

If the corresponding index is equal to the JButton index and board DoublyLinkedList contains a 'b'

Set JButton background to blue

Set JButton text to 'X'

If the corresponding index is equal to the JButton index and board DoublyLinkedList contains a 'w'

Set JButton background to white

PSEUDO CODE (CONT.)

Set JButton text to 'X'

JLabels will be set by counting the number of blue and white squares on the board

Game JLayeredPane will appear while start JLayeredPane will hide

If the player clicks any JButton on the board

ActionListener will check which button was clicked

If the corresponding index of the button is equal to the index of the board DoublyLinkedList and contains character 'G'

Change the button to blue

Set the character from 'G' to 'B'

If the corresponding index of the button is equal to the index of the board DoublyLinkedList and contains character 'B'

Change the button to white

Set the character from 'B' to 'W'

If the corresponding index of the button is equal to the index of the board DoublyLinkedList and contains character 'W'

Change the button to grey

Set the character from 'W' to 'G'

Update the change in color to the board DoublyLinkedList

Update the JLabels new values

If player finished the game

Game JLayeredPane hides, solved JLayeredPane appears

If the another button is clicked

The player is sent back to the beginning screen to choose an easy or hard level

If the done button is clicked

The program will exit

If the player clicks the help button

PSEUDO CODE (CONT.)

The JLabels surrounding the board becomes visible to show the number of blues and whites in each row and column

If the player clicks the help button again

The JLabels surrounding the board becomes invisible

JAVADOC

project1

Class Game_JFrame

java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Frame
          javax.swing.JFrame
            project1.Game_JFrame

All Implemented Interfaces:

`java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants`

`public class Game_JFrame`
`extends javax.swing.JFrame`

See Also:

`Serialized Form`

Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
class	`Game_JFrame.ButtonListener` This method will check which button in the button table was clicked and check which corresponding character in the boardList the button shared to change the color of the button.

Nested classes/interfaces inherited from class javax.swing.JFrame

`javax.swing.JFrame.AccessibleJFrame`

Nested classes/interfaces inherited from class java.awt.Frame

`java.awt.Frame.AccessibleAWTFrame`

Nested classes/interfaces inherited from class java.awt.Window

`java.awt.Window.AccessibleAWTWindow, java.awt.Window.Type`

Constructor Summary

Constructors

Constructor and Description
`Game_JFrame()` Creates new form Game_JFrame

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
`boolean`	`check3InARow()` This method will check that the squares in each row or column do not have three of the same colors in a row.
`boolean`	`checkAnswer()` This method will return true if the board has the same amount of whites and blues in each row and column.
`void`	`countBWInCols()` This method will count the numbers of blues and whites in each column by searching through the boardColList DoublyLinkedList and save the information to colsBW Vector.
`void`	`countBWInRows()` This method will count the numbers of blues and whites in each column by searching through the boardList DoublyLinkedList and save the information to rowsBW Vector.
`static void`	`main(java.lang.String[] args)`
`void`	`updateBoardColList()` This method will change the boardColList DoublyLinkedList corresponding to the boardList DoublyLinkedList within this class.

Constructor Detail

Game_JFrame

```
public Game_JFrame()
```

Creates new form Game_JFrame

Method Detail

updateBoardColList

```
public void updateBoardColList()
```

This method will change the boardColList DoublyLinkedList corresponding to the boardList DoublyLinkedList within this class.

check3InARow

```
public boolean check3InARow()
```

This method will check that the squares in each row or column do not have three of the same colors in a row. It will return true if there is no three in a row found. It will return false if there is a three in a row found.

Returns:

`boolean`

checkAnswer

```
public boolean checkAnswer()
```

This method will return true if the board has the same amount of whites and blues in each row and column. It will return false if there is a three in a row detected.

Returns:

`boolean`

countBWInRows

```
public void countBWInRows()
```

This method will count the numbers of blues and whites in each column by searching through the boardList DoublyLinkedList and save the information to rowsBW Vector.

countBWInCols

```
public void countBWInCols()
```

This method will count the numbers of blues and whites in each column by searching through the boardColList DoublyLinkedList and save the information to colsBW Vector.

main

```
public static void main(java.lang.String[] args)
```

Parameters:

`args - the command line arguments`

project1

Class DoublyLinkedList

java.lang.Object
    project1.DoublyLinkedList

```
public class DoublyLinkedList
extends java.lang.Object
```

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	**changeColor**(java.lang.Integer row, java.lang.Integer column, java.lang.Character color) This function lets the user change the letter in the board by selecting the row and column and then replacing the current color with the color passed in.
void	**clearList**() This method empties the DoublyLinkedList
Link	**deleteHeadLink**() This method deletes the head (very first) Link of list
boolean	**deleteLink**(java.lang.Integer index) This method deletes a link in LinkedList by passing in desired index.
Link	**deleteNewestLink**() This method deletes the newestLink added(or in other terms: the last Link added) to the LinkedList.
java.lang.Character	**findCharacter**(java.lang.Integer row, java.lang.Integer column) This method return the character corresponding to the position of row and column in a 6x6 board.
boolean	**insertAfter**(java.lang.Integer index, java.lang.Integer indexInLinkedList, java.lang.Character color) This method inserts an index after indexInLink ex: insertAfter(3,7) = inserts 3 after 7

void	**insertHeadLink**(java.lang.Integer index, java.lang.Character color) This method adds a Link from the front of the List
void	**insertNewLink**(java.lang.Integer index, java.lang.Character color) This method adds a Link to the List from the end by passing in an index to add
boolean	**isLinkedListEmpty**() This method checks if LinkedList is empty or filled
void	**printLinkedListCol**(int perLine) This method will print the DoublyLinkedList in order of a board rotated to the right
void	**printLinkedListColor**(int perLine) This method will print the characters from beginning and check how many per line to display by passing in an int value.
void	**printLinkedListIndex**(int perLine) This method will print from the first index added and check how many per line to display by passing in an int value.
void	**printLinkedListInReverseOrder**() This method will print from the most recently added index

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

isLinkedListEmpty

```
public boolean isLinkedListEmpty()
```

This method checks if LinkedList is empty or filled

Returns:

boolean

insertHeadLink

```
public void insertHeadLink(java.lang.Integer index,
                           java.lang.Character color)
```

This method adds a Link from the front of the List

Parameters:

index - Integer of newLink

color - Character of newLink

insertNewLink

```
public void insertNewLink(java.lang.Integer index,
                          java.lang.Character color)
```

This method adds a Link to the List from the end by passing in an index to add

Parameters:

index - Integer of newLink

color - Character of newLink

deleteHeadLink

```
public Link deleteHeadLink()
```

This method deletes the head (very first) Link of list

Returns:

Link - headLink

deleteNewestLink

```
public Link deleteNewestLink()
```

This method deletes the newestLink added(or in other terms: the last Link added) to the LinkedList.

Returns:

`Link - newestLink`

clearList

```
public void clearList()
```

This method empties the DoublyLinkedList

insertAfter

```
public boolean insertAfter(java.lang.Integer index,
                           java.lang.Integer indexInLinkedList,
                           java.lang.Character color)
```

This method inserts an index after indexInLink ex: insertAfter(3,7) = inserts 3 after 7

Parameters:

`index - Integer to add to Link to add to DoublyLinkedList`

`indexInLinkedList - value of existent Integer in DoublyLinkedList`

`color - - Character of Link in given index of DoublyLinekdList`

Returns:

`boolean`

changeColor

```
public void changeColor(java.lang.Integer row,
                        java.lang.Integer column,
                        java.lang.Character color)
```

This function lets the user change the letter in the board by selecting the row and column and then replacing the current color with the color passed in.

Parameters:

`row - value of the row in List`

`column - value of column in List`

`color - Character of Link`

findCharacter

```
public java.lang.Character findCharacter(java.lang.Integer row,
                                         java.lang.Integer column)
```

This method return the character corresponding to the position of row and column in a 6x6 board.

Parameters:

`row - value of row in DoublyLinkedList`

`column - value of column DoublyLinkedList`

Returns:

`Character - color of Link`

deleteLink

```
public boolean deleteLink(java.lang.Integer index)
```

This method deletes a link in LinkedList by passing in desired index.

Parameters:

`index - Integer in Link to be deleted`

Returns:

`boolean`

printLinkedListIndex

```
public void printLinkedListIndex(int perLine)
```

This method will print from the first index added and check how many per line to display by passing in an int value.

Parameters:

`perLine - number of displays in a row`

printLinkedListColor

```
public void printLinkedListColor(int perLine)
```

This method will print the characters from beginning and check how many per line to display by passing in an int value.

Parameters:

`perLine - number of displays in a row`

printLinkedListCol

```
public void printLinkedListCol(int perLine)
```

This method will print the DoublyLinkedList in order of a board rotated to the right

Parameters:

`perLine - number of displays in a row`

printLinkedListInReverseOrder

```
public void printLinkedListInReverseOrder()
```

This method will print from the most recently added index

project1

Class Levels

java.lang.Object
    project1.Levels

```
public class Levels
extends java.lang.Object
```

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
java.lang.String	`getEasyLevel()` This method will return an easy level
java.lang.String	`getHardLevel()` This method will return an hard level

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Method Detail

getEasyLevel

```
public java.lang.String getEasyLevel()
```

This method will return an easy level

Returns:
`String`

getHardLevel

```
public java.lang.String getHardLevel()
```

This method will return an hard level

Returns:
`String`

project1

Class Link

java.lang.Object
    project1.Link

```
public class Link
extends java.lang.Object
```

Constructor Summary

Constructors

Constructor and Description
`Link()`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`addLink(java.lang.Integer index, java.lang.Character color)` This method sets the index and color to this Link
`java.lang.Character`	`getCharacter()` This method returns the Link's color Character
`java.lang.Integer`	`getIndex()` This method returns the Link's index Integer
`void`	`printLinkColor()` This method displays the color of this Link
`void`	`printLinkIndex()` This method displays the index of this Link
`void`	`setCharacter(java.lang.Character color)` This method sets the Link's color Character

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Link

```
public Link()
```

Method Detail

addLink

```
public void addLink(java.lang.Integer index,
                    java.lang.Character color)
```

This method sets the index and color to this Link

Parameters:

index - Integer of Link

color - Character of Link

getIndex

```
public java.lang.Integer getIndex()
```

This method returns the Link's index Integer

Returns:

Integer

getCharacter

```
public java.lang.Character getCharacter()
```

This method returns the Link's color Character

Returns:

Character

setCharacter

```
public void setCharacter(java.lang.Character color)
```

This method sets the Link's color Character

Parameters:

color - Character of Link

printLinkIndex

```
public void printLinkIndex()
```

This method displays the index of this Link

printLinkColor

```
public void printLinkColor()
```

This method displays the color of this Link

project1

Class Vector

java.lang.Object
    project1.Vector

```
public class Vector
extends java.lang.Object
```

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	**add**(int number) This method adds a number to the Vector and will double size if needed
boolean	**checkAdd**() This method checks to make sure the index does not go out of bounds.
boolean	**checkNoMuliple**(int num) This method will check for any multiple numbers in the Vector and return true if there is not a duplicate or return false if there is a duplicate
void	**emptyVector**() This method will empty the Vector
int	**getNumber**(int index) This method will return the number at a specific index in the Vector.
int	**getSize**() This method will return the size of this Vector
boolean	**isEmpty**() This method returns true if array is empty or false if it is filled
void	**print**(int perLine) This method will print out the values in the Vector
void	**setArray**(int[] arr) This function sets the passed in array to this Vector's array
void	**setArraySize**(int size) This method sets the size of this Vector
int[]	**sortArray**() This method will sort the array in the Vector and return an int array.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

isEmpty

```
public boolean isEmpty()
```

This method returns true if array is empty or false if it is filled

Returns:
boolean

setArraySize

```
public void setArraySize(int size)
```

This method sets the size of this Vector

Parameters:
size - of Vector

add

```
public void add(int number)
```

This method adds a number to the Vector and will double size if needed

Parameters:

`number - to add to the Vector`

checkAdd

```
public boolean checkAdd()
```

This method checks to make sure the index does not go out of bounds. It will return true if not out of bounds, it will return false if index is out of bounds.

Returns:

`boolean`

sortArray

```
public int[] sortArray()
```

This method will sort the array in the Vector and return an int array.

Returns:

`int[]`

setArray

```
public void setArray(int[] arr)
```

This function sets the passed in array to this Vector's array

Parameters:

`arr - - an integer array`

checkNoMuliple

```
public boolean checkNoMuliple(int num)
```

This method will check for any multiple numbers in the Vector and return true if there is not a duplicate or return false if there is a duplicate

Parameters:

`num - value to check in the array`

Returns:

`boolean`

getNumber

```
public int getNumber(int index)
```

This method will return the number at a specific index in the Vector.

Parameters:

`index - to a number in the Vector`

Returns:

`int - number in the Vector at passed in index`

emptyVector

```
public void emptyVector()
```

This method will empty the Vector

getSize

```
public int getSize()
```

This method will return the size of this Vector

Returns:

```
int - size of Vector
```

print

```
public void print(int perLine)
```

This method will print out the values in the Vector

Parameters:

```
perLine - number to display in a row
```

REFERENCES

Java How To Program (Late objects) 10th Edition by Deitel

http://stackoverflow.com/

CS 61B Lecture 7: Linked Lists I

PROGRAM

```
/*
 * File: Project 1
 * Programmer: Jasmine Anica
 * Class: CSC 18C
 * Date: 5/20/15
 */
package project1;

import java.awt.Color;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BorderFactory;
import javax.swing.JButton;


/**
 *
 * @author Jasmine
 */
public class Game_JFrame extends javax.swing.JFrame {

    DoublyLinkedList boardList = new DoublyLinkedList();
    DoublyLinkedList boardColList = new DoublyLinkedList();
    Vector rowsBW = new Vector();
    Vector colsBW = new Vector();
    Levels levels = new Levels();

    Color blue = new Color(0,118,153);
    Font font0 = new Font("Lucida sans", Font.BOLD, 20);

    private JButton[][] buttons;
    private int a = 6;



    /**
     * Creates new form Game_JFrame
     */
    public Game_JFrame() {
        initComponents();
        setRowColLabelInvisible();
        setLocationRelativeTo(null);


        //initializes a Linkedlist with the columns
        for (int i = 1; i <= 6; i++){
            boardColList.insertNewLink(i, 'G');
            for (int j = 7; j <= 36; j+=6){
                boardColList.insertNewLink(i+(j-1), 'G');
            }
```

```
        }

    }

    /**
     * This method is called from within the constructor to initialize the
form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {

        solved = new javax.swing.JLayeredPane();
        solvedInternalFrame = new javax.swing.JInternalFrame();
        winnerIconLabel = new javax.swing.JLabel();
        puzzleSolvedLabel = new javax.swing.JLabel();
        doneButton = new javax.swing.JToggleButton();
        anotherButton = new javax.swing.JToggleButton();
        startScreen = new javax.swing.JLayeredPane();
        easyButton = new javax.swing.JButton();
        hardButton = new javax.swing.JButton();
        gameTitle = new javax.swing.JLabel();
        game = new javax.swing.JLayeredPane();
        gameTitle1 = new javax.swing.JLabel();
        board = new javax.swing.JLayeredPane();
        row1Label = new javax.swing.JLabel();
        row2Label = new javax.swing.JLabel();
        row3Label = new javax.swing.JLabel();
        row4Label = new javax.swing.JLabel();
        row5Label = new javax.swing.JLabel();
        row6Label = new javax.swing.JLabel();
        col1Label = new javax.swing.JLabel();
        col2Label = new javax.swing.JLabel();
        col3Label = new javax.swing.JLabel();
        col4Label = new javax.swing.JLabel();
        col5Label = new javax.swing.JLabel();
        col6Label = new javax.swing.JLabel();
        helpModeButton = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("3-in-a-row");
        setBackground(new java.awt.Color(204, 255, 255));
        setMaximumSize(new java.awt.Dimension(900, 800));
        setMinimumSize(new java.awt.Dimension(900, 800));
        setResizable(false);
        getContentPane().setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());

        solved.setVisible(false);
        solved.setBackground(new java.awt.Color(0, 118, 153));
```

```
        solved.setForeground(new java.awt.Color(0, 118, 153));
        solved.setOpaque(true);
        solved.setPreferredSize(new java.awt.Dimension(900, 800));


solvedInternalFrame.setBorder(javax.swing.BorderFactory.createBevelBorder(jav
ax.swing.border.BevelBorder.RAISED));
        solvedInternalFrame.setTitle("Solved");
        solvedInternalFrame.setVisible(true);
        solvedInternalFrame.getContentPane().setLayout(null);

        winnerIconLabel.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/project1/completed_ribbon_100x
100.png"))); // NOI18N
        solvedInternalFrame.getContentPane().add(winnerIconLabel);
        winnerIconLabel.setBounds(50, 60, 100, 100);

        puzzleSolvedLabel.setFont(new java.awt.Font("Lucida Sans", 0, 48));
// NOI18N
        puzzleSolvedLabel.setForeground(new java.awt.Color(0, 118, 153));
        puzzleSolvedLabel.setText("Puzzle Solved");
        solvedInternalFrame.getContentPane().add(puzzleSolvedLabel);
        puzzleSolvedLabel.setBounds(190, 70, 370, 57);

        doneButton.setFont(new java.awt.Font("Lucida Sans", 0, 18)); //
NOI18N
        doneButton.setForeground(new java.awt.Color(0, 118, 153));
        doneButton.setText("Done");
        doneButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                doneButtonMouseClicked(evt);
            }
        });
        solvedInternalFrame.getContentPane().add(doneButton);
        doneButton.setBounds(390, 160, 100, 31);

        anotherButton.setFont(new java.awt.Font("Lucida Sans", 0, 18)); //
NOI18N
        anotherButton.setForeground(new java.awt.Color(0, 118, 153));
        anotherButton.setText("Another");
        anotherButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                anotherButtonMouseClicked(evt);
            }
        });
        solvedInternalFrame.getContentPane().add(anotherButton);
        anotherButton.setBounds(230, 160, 119, 31);

        javax.swing.GroupLayout solvedLayout = new
javax.swing.GroupLayout(solved);
        solved.setLayout(solvedLayout);
        solvedLayout.setHorizontalGroup(

solvedLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
solvedLayout.createSequentialGroup()
                .addContainerGap(161, Short.MAX_VALUE)
```

```
                .addComponent(solvedInternalFrame,
javax.swing.GroupLayout.PREFERRED_SIZE, 612,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(127, 127, 127))
        );
        solvedLayout.setVerticalGroup(

solvedLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(solvedLayout.createSequentialGroup()
                .addGap(141, 141, 141)
                .addComponent(solvedInternalFrame,
javax.swing.GroupLayout.PREFERRED_SIZE, 280,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(379, Short.MAX_VALUE))
        );
        solved.setLayer(solvedInternalFrame,
javax.swing.JLayeredPane.DEFAULT_LAYER);

        getContentPane().add(solved, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, -1, -1));

        startScreen.setBackground(new java.awt.Color(0, 118, 153));
        startScreen.setOpaque(true);
        startScreen.setPreferredSize(new java.awt.Dimension(900, 800));
        startScreen.setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());

        easyButton.setBackground(new java.awt.Color(255, 255, 255));
        easyButton.setFont(new java.awt.Font("Lucida Sans", 1, 36)); //
NOI18N
        easyButton.setForeground(new java.awt.Color(0, 118, 153));
        easyButton.setText("Easy Mode");

easyButton.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.
border.BevelBorder.RAISED));
        easyButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                easyButtonMouseClicked(evt);
            }
        });
        startScreen.add(easyButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(130, 430, 260, 70));

        hardButton.setBackground(new java.awt.Color(255, 255, 255));
        hardButton.setFont(new java.awt.Font("Lucida Sans", 1, 36)); //
NOI18N
        hardButton.setForeground(new java.awt.Color(0, 118, 153));
        hardButton.setText("Hard Mode");

hardButton.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.
border.BevelBorder.RAISED));
        hardButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                hardButtonMouseClicked(evt);
            }
        });
```

```
        startScreen.add(hardButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(520, 430, 260, 70));

        gameTitle.setFont(new java.awt.Font("Lucida Sans", 3, 48)); // NOI18N
        gameTitle.setForeground(new java.awt.Color(255, 255, 255));
        gameTitle.setText("3 - in - a - row");
        gameTitle.setBorder(javax.swing.BorderFactory.createLineBorder(new
java.awt.Color(255, 255, 255), 4));
        startScreen.add(gameTitle, new
org.netbeans.lib.awtextra.AbsoluteConstraints(240, 150, 410, 70));

        getContentPane().add(startScreen, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, -1, -1));

        game.setVisible(false);
        game.setBackground(new java.awt.Color(0, 118, 153));
        game.setToolTipText("");
        game.setOpaque(true);
        game.setPreferredSize(new java.awt.Dimension(900, 800));
        game.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        gameTitle1.setFont(new java.awt.Font("Lucida Sans", 3, 48)); //
NOI18N
        gameTitle1.setForeground(new java.awt.Color(255, 255, 255));
        gameTitle1.setText("3 - in - a - row");
        game.add(gameTitle1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(260, 10, 410, 70));

        board.setBackground(new java.awt.Color(204, 204, 255));
        board.setPreferredSize(new java.awt.Dimension(500, 500));

        javax.swing.GroupLayout boardLayout = new
javax.swing.GroupLayout(board);
        board.setLayout(boardLayout);
        boardLayout.setHorizontalGroup(

boardLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 500, Short.MAX_VALUE)
        );
        boardLayout.setVerticalGroup(

boardLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 500, Short.MAX_VALUE)
        );

        game.add(board, new
org.netbeans.lib.awtextra.AbsoluteConstraints(195, 117, -1, -1));

        row1Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        row1Label.setForeground(new java.awt.Color(255, 255, 255));
        row1Label.setText("0/6");
        row1Label.setMaximumSize(new java.awt.Dimension(40, 20));
        game.add(row1Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(710, 140, 100, 40));

        row2Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        row2Label.setForeground(new java.awt.Color(255, 255, 255));
```

```
        row2Label.setText("0/6");
        game.add(row2Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(710, 230, 100, -1));

        row3Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        row3Label.setForeground(new java.awt.Color(255, 255, 255));
        row3Label.setText("0/6");
        row3Label.setToolTipText("");
        game.add(row3Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(710, 300, 100, 40));

        row4Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        row4Label.setForeground(new java.awt.Color(255, 255, 255));
        row4Label.setText("0/6");
        game.add(row4Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(710, 380, 100, 40));

        row5Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        row5Label.setForeground(new java.awt.Color(255, 255, 255));
        row5Label.setText("0/6");
        game.add(row5Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(710, 460, 100, 40));

        row6Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        row6Label.setForeground(new java.awt.Color(255, 255, 255));
        row6Label.setText("0/6");
        game.add(row6Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(710, 540, 100, 40));

        col1Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        col1Label.setForeground(new java.awt.Color(255, 255, 255));
        col1Label.setText("0/6");
        col1Label.setMaximumSize(new java.awt.Dimension(40, 20));
        game.add(col1Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(220, 80, -1, -1));

        col2Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        col2Label.setForeground(new java.awt.Color(255, 255, 255));
        col2Label.setText("0/6");
        col2Label.setMaximumSize(new java.awt.Dimension(40, 20));
        game.add(col2Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(300, 80, -1, -1));

        col3Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        col3Label.setForeground(new java.awt.Color(255, 255, 255));
        col3Label.setText("0/6");
        col3Label.setMaximumSize(new java.awt.Dimension(40, 20));
        game.add(col3Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(380, 80, -1, -1));

        col4Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        col4Label.setForeground(new java.awt.Color(255, 255, 255));
        col4Label.setText("0/6");
        col4Label.setMaximumSize(new java.awt.Dimension(40, 20));
        game.add(col4Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(460, 80, -1, -1));
```

```
        col5Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        col5Label.setForeground(new java.awt.Color(255, 255, 255));
        col5Label.setText("0/6");
        col5Label.setMaximumSize(new java.awt.Dimension(40, 20));
        game.add(col5Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(540, 80, -1, -1));

        col6Label.setFont(new java.awt.Font("Lucida Sans", 1, 24)); // NOI18N
        col6Label.setForeground(new java.awt.Color(255, 255, 255));
        col6Label.setText("0/6");
        col6Label.setMaximumSize(new java.awt.Dimension(40, 20));
        game.add(col6Label, new
org.netbeans.lib.awtextra.AbsoluteConstraints(630, 80, -1, -1));

        helpModeButton.setBackground(new java.awt.Color(255, 255, 255));
        helpModeButton.setFont(new java.awt.Font("Lucida Sans", 1, 18)); //
NOI18N
        helpModeButton.setForeground(new java.awt.Color(0, 102, 153));
        helpModeButton.setText("Help Mode");
        helpModeButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                helpModeButtonMouseClicked(evt);
            }
        });
        game.add(helpModeButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(710, 640, 150, 30));

        jLabel2.setFont(new java.awt.Font("Lucida Sans", 1, 18)); // NOI18N
        jLabel2.setForeground(new java.awt.Color(255, 255, 255));
        jLabel2.setText(">Each row and column must have an EQUAL number of
Blue and White squares.");
        game.add(jLabel2, new
org.netbeans.lib.awtextra.AbsoluteConstraints(60, 750, -1, -1));

        jLabel3.setFont(new java.awt.Font("Lucida Sans", 3, 24)); // NOI18N
        jLabel3.setForeground(new java.awt.Color(255, 255, 255));
        jLabel3.setText("Objective / Rules");
        jLabel3.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        game.add(jLabel3, new
org.netbeans.lib.awtextra.AbsoluteConstraints(130, 640, -1, -1));

        jLabel4.setFont(new java.awt.Font("Lucida Sans", 1, 18)); // NOI18N
        jLabel4.setForeground(new java.awt.Color(255, 255, 255));
        jLabel4.setText(">Fill the grid with Blue and White squares.");
        game.add(jLabel4, new
org.netbeans.lib.awtextra.AbsoluteConstraints(60, 690, -1, -1));

        jLabel5.setFont(new java.awt.Font("Lucida Sans", 1, 18)); // NOI18N
        jLabel5.setForeground(new java.awt.Color(255, 255, 255));
        jLabel5.setText(">A 3-In-A-Row of the same color is NOT allowed.");
        game.add(jLabel5, new
org.netbeans.lib.awtextra.AbsoluteConstraints(60, 720, -1, -1));

        getContentPane().add(game, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, -1, 800));

        pack();
```

```
    }// </editor-fold>//GEN-END:initComponents

    /**
     * This method is called when help mode JButton is clicked. It calls the
     * function used to display or hide the JLabels containing the number of
     * blues and whites in each row and column.
     *
     * @param evt
     */
    private void helpModeButtonMouseClicked(java.awt.event.MouseEvent evt)
{//GEN-FIRST:event_helpModeButtonMouseClicked

        if (helpModeButton.getBackground() == Color.WHITE){
            setRowColLabelVisible();
            helpModeButton.setForeground(Color.WHITE);
            helpModeButton.setBackground(blue);
        } else {

            setRowColLabelInvisible();
            helpModeButton.setForeground(blue);
            helpModeButton.setBackground(Color.WHITE);
        }
    }//GEN-LAST:event_helpModeButtonMouseClicked

/**
 * When easy JButton is clicked, it will close the current JLayeredPane and
display
 * the game JLayeredPane. This also includes generating a new table for the
board and
 * setting up the JLabels and JButtons.
 *
 * @param evt
 */
    private void easyButtonMouseClicked(java.awt.event.MouseEvent evt)
{//GEN-FIRST:event_easyButtonMouseClicked
        // TODO add your handling code here:

        boardList.clearList();
        String easyLvl = levels.getEasyLevel();


        //creates a board LinkedList
        for (int i = 1; i <= a*a; i++) {
            boardList.insertNewLink(i, easyLvl.charAt(i - 1));
        }

        updateBoardColList();
        ButtonTable();
        setFrozen();
        setRowNumberLabels();
        setColNumberLabels();

        startScreen.setVisible(false);
        game.setVisible(true);

    }//GEN-LAST:event_easyButtonMouseClicked
```

```
    /**
     * When hard JButton is clicked, this method will will close the current
     * JLayeredPane and display the game JLayeredPane. This also includes
     * generating a new table for the board and setting up the JLabels and
JButtons.
     *
     * @param evt
     */
    private void hardButtonMouseClicked(java.awt.event.MouseEvent evt)
{//GEN-FIRST:event_hardButtonMouseClicked

        boardList.clearList();

        String hardLvl = levels.getHardLevel();

        //creates a board LinkedList
        for (int i = 1; i <= a*a; i++) {
            boardList.insertNewLink(i, hardLvl.charAt(i - 1));
        }

        updateBoardColList();
        ButtonTable();
        setFrozen();
        setRowNumberLabels();
        setColNumberLabels();

        startScreen.setVisible(false);
        game.setVisible(true);
    }//GEN-LAST:event_hardButtonMouseClicked

    /**
     * This method will close the program when done JButton is clicked.
     *
     * @param evt
     */
    private void doneButtonMouseClicked(java.awt.event.MouseEvent evt)
{//GEN-FIRST:event_doneButtonMouseClicked
    System.exit(0);
    }//GEN-LAST:event_doneButtonMouseClicked

/**
 * When another JButton is clicked, the game will start the startScreen
 * JLayeredPane to restart at the beginning.
 *
 * @param evt
 */
    private void anotherButtonMouseClicked(java.awt.event.MouseEvent evt)
{//GEN-FIRST:event_anotherButtonMouseClicked
        // TODO add your handling code here:
        solved.setVisible(false);
        startScreen.setVisible(true);
    }//GEN-LAST:event_anotherButtonMouseClicked

       /**
     * This method will change the boardColList DoublyLinkedList
corresponding
     * to the boardList DoublyLinkedList within this class.
```

```
     */
    public void updateBoardColList(){

        Link current = boardList.headLink;

        while(current != null){

            Link currentCol = boardColList.headLink;
            while(currentCol != null){

                if(current.getIndex().equals(currentCol.getIndex())){
                    currentCol.setCharacter(current.getCharacter());
                }

            currentCol = currentCol.next;
            }
            current = current.next;
        }
    }


    /**
     * This method will check that the squares in each row or column do not
     * have three of the same colors in a row. It will return true if there
is no
     * three in a row found. It will return false if there is a three in a
row found.
     *
     * @return boolean
     */
    public boolean check3InARow(){

        Link current = boardList.headLink;
        Integer number;


        while(current.next.next != null){

            number = current.getIndex() % 6;

            //does not include the last two in each row
            if(number != 0 && number <=4 && current.getCharacter() != 'G'){

                Character currentChar =
Character.toUpperCase(current.getCharacter());
                Character nextChar =
Character.toUpperCase(current.next.getCharacter());
                Character nextNextChar =
Character.toUpperCase(current.next.next.getCharacter());

                //check the current's neighbor and neighbor's neighbor are
the same character
                if(currentChar.equals(nextChar) &&
                    nextChar.equals(nextNextChar)){
                    System.out.println("Mistake: A three in a Row (in a
row)");
                    return false; // there is a three in a row
```

```
                }
            }

            current = current.next;
        }

        Link current2 = boardColList.headLink;

        int counter = 0;

        while(current2.next.next != null){

            counter++;
            number = counter % 6;

            //does not include the last two in each col
            if(number != 0 && number <=4 && current2.getCharacter() != 'G'){

                Character current2Char =
Character.toUpperCase(current2.getCharacter());
                Character nextChar =
Character.toUpperCase(current2.next.getCharacter());
                Character nextNextChar =
Character.toUpperCase(current2.next.next.getCharacter());

                //check the current's neighbor and neighbor's neighbor are
the same character
                if(current2Char.equals(nextChar) &&
                    nextChar.equals(nextNextChar)){
                    System.out.println("Mistake: A three in a Row (in a
col)");
                    return false; // there is a three in a row
                }
            }

            current2 = current2.next;
        }
        return true; // no three in a rows found

    }



    /**
     * This method will return true if the board has the same amount of
whites
     * and blues in each row and column. It will return false if there is a
three
     * in a row detected.
     *
     * @return boolean
     */
    public boolean checkAnswer(){

        for(int i = 0; i < 12; i++){

            //if the index is even
```

```
            if(i % 2 == 0){
                if(rowsBW.getNumber(0) != rowsBW.getNumber(i) ||
                        colsBW.getNumber(0) != colsBW.getNumber(i) ||
                        rowsBW.getNumber(0) != colsBW.getNumber(i)){
                    return false;
                }

            } else { //if index is odd
                if (rowsBW.getNumber(1) != rowsBW.getNumber(i) ||
                        colsBW.getNumber(1) != colsBW.getNumber(i) ||
                        rowsBW.getNumber(1) != colsBW.getNumber(i)){
                    return false;
                }
            }
        }

        return true;
    }



    /**
     * This method will count the numbers of blues and whites in each column
     * by searching through the boardList DoublyLinkedList and save the
     * information to rowsBW Vector.
     */
    public void countBWInRows(){

        Link current = boardList.headLink;
        //boardList.printLinkedListColor(6);

        int counter = 1;
        int blues = 0;
        int whites = 0;

        while(current != null){

            if (current.getCharacter() == 'B' || current.getCharacter() ==
'b'){
                blues++;
            } else if (current.getCharacter() == 'W' ||
current.getCharacter() == 'w'){
                whites++;
            }

            if (counter % 6 == 0){
                rowsBW.add(blues);
                rowsBW.add(whites);
                blues = 0;
                whites = 0;
            }

            current = current.next;
            counter++;
        }

        //rowsBW.print(6);
```

```
    }

    /**
     * This method will count the numbers of blues and whites in each column
     * by searching through the boardColList DoublyLinkedList and save the
     * information to colsBW Vector.
     */
    public void countBWInCols(){
        Link current = boardColList.headLink;
        //boardColList.printLinkedListCol(6);

        int counter = 1;
        int blues = 0;
        int whites = 0;

        while(current != null){

            if (current.getCharacter() == 'B' || current.getCharacter() ==
'b'){
                blues++;
            } else if (current.getCharacter() == 'W' ||
current.getCharacter() == 'w'){
                whites++;

            }

            if (counter % 6 == 0){
                colsBW.add(blues);
                colsBW.add(whites);
                blues = 0;
                whites = 0;
            }

            current = current.next;
            counter++;
        }

        //colsBW.print(6);
    }

    /**
     * This method will search through the boardList DoublyLinkedList to
locate
     * a character 'b' or a 'w'. If it finds either, the button will have a
mark
     * 'x' to show the player the button is fixed (cannot be changed) to the
board.
     */
    private void setFrozen(){

        for (int i = 0; i < a; i++){
            for (int j = 0; j < a; j++){
                if(boardList.findCharacter(i + 1, j + 1) == 'b'){
                    buttons[i][j].setBackground(blue);
                    buttons[i][j].setFont(font0);
                    buttons[i][j].setForeground(Color.WHITE);
                    buttons[i][j].setText("X");
```

```
            } else if(boardList.findCharacter(i + 1, j + 1) == 'w'){
                buttons[i][j].setBackground(Color.WHITE);
                buttons[i][j].setFont(font0);
                buttons[i][j].setForeground(blue);
                buttons[i][j].setText("X");
            }
        }
    }

}


/**
 * This method will open solved JLayeredPane to show the player they have
 * successfully solved the puzzle.
 */
private void finished(){
    if(checkAnswer() && check3InARow()){
        game.setVisible(false);
        solved.setVisible(true);

        System.out.println("Completed");
    }
}

/**
 * This method displays the number of blues and whites in each row using
 * JLabels.
 */
private void setRowNumberLabels(){
    countBWInRows();

    String blue;
    String white;

    blue = String.valueOf(rowsBW.getNumber(0));
    white = String.valueOf(rowsBW.getNumber(1));
    row1Label.setText(blue+"/"+white);

    blue = String.valueOf(rowsBW.getNumber(2));
    white = String.valueOf(rowsBW.getNumber(3));
    row2Label.setText(blue+"/"+white);

    blue = String.valueOf(rowsBW.getNumber(4));
    white = String.valueOf(rowsBW.getNumber(5));
    row3Label.setText(blue+"/"+white);

    blue = String.valueOf(rowsBW.getNumber(6));
    white = String.valueOf(rowsBW.getNumber(7));
    row4Label.setText(blue+"/"+white);

    blue = String.valueOf(rowsBW.getNumber(8));
    white = String.valueOf(rowsBW.getNumber(9));
    row5Label.setText(blue+"/"+white);

    blue = String.valueOf(rowsBW.getNumber(10));
    white = String.valueOf(rowsBW.getNumber(11));
```

```
        row6Label.setText(blue+"/"+white);

    }

    /**
     * This method displays the number of blues and whites in each column
using
     * JLabels.
     */
    private void setColNumberLabels(){
        countBWInCols();

        String blue;
        String white;

        blue = String.valueOf(colsBW.getNumber(0));
        white = String.valueOf(colsBW.getNumber(1));
        col1Label.setText(blue+"/"+white);

        blue = String.valueOf(colsBW.getNumber(2));
        white = String.valueOf(colsBW.getNumber(3));
        col2Label.setText(blue+"/"+white);

        blue = String.valueOf(colsBW.getNumber(4));
        white = String.valueOf(colsBW.getNumber(5));
        col3Label.setText(blue+"/"+white);

        blue = String.valueOf(colsBW.getNumber(6));
        white = String.valueOf(colsBW.getNumber(7));
        col4Label.setText(blue+"/"+white);

        blue = String.valueOf(colsBW.getNumber(8));
        white = String.valueOf(colsBW.getNumber(9));
        col5Label.setText(blue+"/"+white);

        blue = String.valueOf(colsBW.getNumber(10));
        white = String.valueOf(colsBW.getNumber(11));
        col6Label.setText(blue+"/"+white);


    }

    /**
     * This method sets all the labels displaying the number of blues and
whites
     * in each row and column invisible to the player.
     */
    private void setRowColLabelInvisible(){

        row1Label.setVisible(false);
        row2Label.setVisible(false);
        row3Label.setVisible(false);
        row4Label.setVisible(false);
        row5Label.setVisible(false);
        row6Label.setVisible(false);

        col1Label.setVisible(false);
```

```
        col2Label.setVisible(false);
        col3Label.setVisible(false);
        col4Label.setVisible(false);
        col5Label.setVisible(false);
        col6Label.setVisible(false);

    }

    /**
     * This method sets all the labels displaying the number of blues and
whites
     * in each row and column visible to the player.
     */
    private void setRowColLabelVisible(){

        row1Label.setVisible(true);
        row2Label.setVisible(true);
        row3Label.setVisible(true);
        row4Label.setVisible(true);
        row5Label.setVisible(true);
        row6Label.setVisible(true);

        col1Label.setVisible(true);
        col2Label.setVisible(true);
        col3Label.setVisible(true);
        col4Label.setVisible(true);
        col5Label.setVisible(true);
        col6Label.setVisible(true);

    }


    /**
     * This method will create a GridLayout of buttons and initialize their
     * appearance and add them to board JLayeredPane.
     */
    private void ButtonTable() {

        board.removeAll();

        //adds a buttonListener
        ActionListener clicked = new ButtonListener();

        //adds grid to place 2D array
        board.setLayout(new GridLayout(a,a));

        //declare size of button 2D array
        buttons = new JButton[a][a];



        for (int i = 0; i < a; i++){
            for (int j = 0; j < a; j++){

                //create new JButton
                buttons[i][j] = new JButton();
```

```
//             buttons_Player[i][j].setText(Integer.toString(i)+"
"+Integer.toString(j));

buttons[i][j].setBorder(BorderFactory.createLineBorder(Color.WHITE, 1));
                buttons[i][j].setOpaque(true);
                buttons[i][j].setBackground(Color.GRAY);
                buttons[i][j].addActionListener(clicked);

                //adds buttons to the layered pane
                board.add(buttons[i][j]);
            }
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting
code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with
the default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Game_JFrame.class.getName()).log(java.util
.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Game_JFrame.class.getName()).log(java.util
.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Game_JFrame.class.getName()).log(java.util
.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Game_JFrame.class.getName()).log(java.util
.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
```

```
                new Game_JFrame().setVisible(true);
            }
        });
    }



    /**
     * This method will check which button in the button table was clicked
and
     * check which corresponding character in the boardList the button shared
to
     * change the color of the button. Then it will call the functions to
check
     * the player has solved the puzzle.
     */
    public class ButtonListener implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e){

            //searches which button was clicked
            for (int i = 0; i < a; i++){
                for (int j = 0; j < a; j++){
                    if (e.getSource() == buttons[i][j]){ //gameButtons[i][j]
was clicked
                        if(boardList.findCharacter(i + 1, j + 1) != 'b' ||
                                boardList.findCharacter(i + 1, j + 1) !=
'w'){


                            //compares to boardList and changes color
                            if (boardList.findCharacter(i + 1, j + 1) ==
'W'){


                                //changes button color
                                buttons[i][j].setBackground(blue);

                                //updates boardList
                                boardList.changeColor(i + 1, j + 1, 'B');
                                updateBoardColList();

                                setRowNumberLabels();
                                setColNumberLabels();
                                finished();

                                rowsBW.emptyVector();
                                colsBW.emptyVector();


                            } else if(boardList.findCharacter(i + 1, j + 1)
== 'B') {


                                //changes button color
                                buttons[i][j].setBackground(Color.GRAY);

                                //updates boardList
                                boardList.changeColor(i + 1, j + 1, 'G');
```

```
                                updateBoardColList();

                                setRowNumberLabels();
                                setColNumberLabels();
                                finished();

                                rowsBW.emptyVector();
                                colsBW.emptyVector();

                            } else if (boardList.findCharacter(i + 1, j + 1)
== 'G'){


                                //changes button color
                                buttons[i][j].setBackground(Color.WHITE);

                                //updates boardList
                                boardList.changeColor(i + 1, j + 1, 'W');
                                updateBoardColList();

                                setRowNumberLabels();
                                setColNumberLabels();
                                finished();

                                rowsBW.emptyVector();
                                colsBW.emptyVector();
                            }

                           check3InARow();
                        }
                    }
                }
            }

      }

    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JToggleButton anotherButton;
    private javax.swing.JLayeredPane board;
    private javax.swing.JLabel col1Label;
    private javax.swing.JLabel col2Label;
    private javax.swing.JLabel col3Label;
    private javax.swing.JLabel col4Label;
    private javax.swing.JLabel col5Label;
    private javax.swing.JLabel col6Label;
    private javax.swing.JToggleButton doneButton;
    private javax.swing.JButton easyButton;
    private javax.swing.JLayeredPane game;
    private javax.swing.JLabel gameTitle;
    private javax.swing.JLabel gameTitle1;
    private javax.swing.JButton hardButton;
    private javax.swing.JButton helpModeButton;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
```

```
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel puzzleSolvedLabel;
    private javax.swing.JLabel row1Label;
    private javax.swing.JLabel row2Label;
    private javax.swing.JLabel row3Label;
    private javax.swing.JLabel row4Label;
    private javax.swing.JLabel row5Label;
    private javax.swing.JLabel row6Label;
    private javax.swing.JLayeredPane solved;
    private javax.swing.JInternalFrame solvedInternalFrame;
    private javax.swing.JLayeredPane startScreen;
    private javax.swing.JLabel winnerIconLabel;
    // End of variables declaration//GEN-END:variables
}


/*
 * File: Project 1
 * Programmer: Jasmine Anica
 * Class: CSC 18C
 * Date: 5/20/15
 */
package project1;

/**
 *
 * @author Jasmine
 */
public class DoublyLinkedList {
    //remembers the first link ONLY
    Link headLink;

    //Link reference that will point to newestLink added (or last link)
    Link newestLink;

    //Constructor
    DoublyLinkedList() {
        newestLink = null;
        headLink = null;
    }

    /**
     * This method checks if LinkedList is empty or filled
     *
     * @return boolean
     */
    public boolean isLinkedListEmpty() {
        return newestLink == null || headLink == null;
    }


   /**
    * This method adds a Link from the front of the List
    *
    * @param index Integer of newLink
    * @param color Character of newLink
    */
```

```
public void insertHeadLink(Integer index , Character color) {

    //Ex: add 3 to front of
    //LinkedList: 5, 2, 7

    //in ex. above newLink = 3
    Link newLink = new Link();
    newLink.addLink(index, color);


    if (isLinkedListEmpty()) { //check for empty list
        newestLink = newLink; //updates newLink
    }
    else {
        // in ex. above headLink = 5
        //sets current headlink.previous = 3
        headLink.previous = newLink;
    }

    //newLink = 3
    //makes newLink.next = 5
    newLink.next = headLink;

    headLink = newLink; //updates headLink to the added Link
}

/**
 * This method adds a Link to the List from the end by passing
 * in an index to add
 *
 * @param index Integer of newLink
 * @param color Character of newLink
 */
public void insertNewLink(Integer index , Character color){

    //Ex: add 8 to
    //List: 6, 4, 12, 1

    //in ex. above newLink = 8
    Link newLink = new Link();
    newLink.addLink(index, color);

    if (isLinkedListEmpty()) { //check is link is empty
        headLink = newLink;
    } else { //if there is a newestLink
        //newestLink = 1
        newestLink.next = newLink; // sets the reference to the newLink
        //we want newLink.previous = 1
        newLink.previous = newestLink;
    }
    newestLink = newLink; //updates the newestLink

    //List: 6, 4, 12, 1, 8
}
```

```
    /**
     * This method deletes the head (very first) Link of list
     *
     * @return Link - headLink
     */
    public Link deleteHeadLink(){

        // Example:
        // List: 1, 2, 3, 4

        //headLink.next above is 2
        //headLink is the first Link in the LinkedList
        if (headLink.next == null) { //true if headLink is the only link
            newestLink = null; //there is no newestLink
        } else {
            //headLink.next = 2
            //headLink.next.previous = 1 but we want to delete that reference
            headLink.next.previous = null;
        }

        //makes headlink now 2
        headLink = headLink.next;

        //List is now: 2, 3, 4,

        return headLink;
    }

    /**
     * This method deletes the newestLink added(or in other terms: the last
Link
     * added) to the LinkedList.
     *
     * @return Link - newestLink
     */
    public Link deleteNewestLink() {

        // Example:
        // List: 1, 2, 3, 4

        //headLink.next above is 2
        //headLink is the first Link in the LinkedList
        if (headLink.next == null) { //true if headLink is the only link
            headLink = null; //deletes newestLink (which happens to be the
head)
        } else {
            //in ex. above, newestLink = 4
            //newestLink.previous = 3
            //newestLink.previous.next = 4 but we want to delete this
reference
            newestLink.previous.next = null;
        }
        //makes newestestLink = 3
        newestLink = newestLink.previous;

        //list is now: 1, 2, 3
```

```
        return newestLink;
    }

    /**
     * This method empties the DoublyLinkedList
     */
    public void clearList(){
        newestLink = null;
        headLink = null;
    }

    /**
     * This method inserts an index after indexInLink
     *
     * ex: insertAfter(3,7) = inserts 3 after 7
     *
     * @param index Integer to add to Link to add to DoublyLinkedList
     * @param indexInLinkedList value of existent Integer in DoublyLinkedList
     * @param color - Character of Link in given index of DoublyLinekdList
     * @return boolean
     */
    public boolean insertAfter(Integer index, Integer indexInLinkedList ,
Character color){
        // Example:
        // add 5 after 8 in
        // List: 3, 8, 7, 6

        //current starts at 3 in example above
        //current begins at the start of the LinkedList
        Link current = headLink;

        //Check to see if the index exists in the LinkedList using current
        while (current.getIndex() != indexInLinkedList) { //breaks when
current = indexInLinkedList

            current = current.next;

            //if current hits the end
            if(current == null) {
                System.out.println("Index in LinkedList not found");
                return false;
            }
        }

        //newLink = 5;
        Link newLink = new Link();
        newLink.addLink(index, color);

        //current is 8 in example above
        //current now equals indexInLinkedList
        if (current == newestLink) {
            newLink.next = null; // newLink is the newestLink so no next
available
            newestLink = newLink; //updates the newestLink with newLink
        } else {

            // List: 3, 8, 7, 6
```

```
            // current = 8, current.next = 7
            // newLink = 5;

            //makes newLink.next = 7
            newLink.next = current.next;
            //makes (current.next(= 7).previous = 8) = 5
            current.next.previous = newLink;

        }
        //makes newLink.prevous = 8
        newLink.previous = current;
        //makes current.next = 5
        current.next = newLink;
        // now: 3, 8, 5, 7 ,6

        return true;
    }

    /**
     *
     * This function lets the user change the letter in the board
     * by selecting the row and column and then replacing the current color
with
     * the color passed in.
     *
     * @param row value of the row in List
     * @param column value of column in List
     * @param color Character of Link
     */
    public void changeColor(Integer row, Integer column, Character color){

        int index = 1;

        //check is list is empty
        if (isLinkedListEmpty()){
            System.out.println("LinkedList is empty. Cannot delete index");
        }

        if (row != 1){
            index += 6*(row - 1);
        }

        //calculated the index
        index += column - 1;

        Link current = headLink;

        while(current.getIndex() != index){

            current = current.next;

            if (current == null) {
                System.out.println("Index not found in LinkedList");
            }

        }
```

```
        current.setCharacter(color);

    }

    /**
     * This method return the character corresponding to the position of row
and
     * column in a 6x6 board.
     *
     * @param row value of row in DoublyLinkedList
     * @param column value of column DoublyLinkedList
     * @return Character - color of Link
     */
    public Character findCharacter(Integer row, Integer column){
        int index = 1;

        //check is list is empty
        if (isLinkedListEmpty()){
            System.out.println("LinkedList is empty. Cannot delete index");
        }

        if (row != 1){
            index += 6*(row - 1);
        }

        //calculated the index
        index += column - 1;

        Link current = headLink;

        while(current.getIndex() != index){

            current = current.next;

            if (current == null) {
                System.out.println("Index not found in LinkedList");
            }

        }

        return current.getCharacter();
    }

    /**
     * This method deletes a link in LinkedList by passing in desired index.
     *
     * @param index Integer in Link to be deleted
     * @return boolean
     */
    public boolean deleteLink(Integer index){

        if (isLinkedListEmpty()){
            System.out.println("LinkedList is empty. Cannot delete index");
            return false;
        }

        Link current = headLink;
```

```
        while(current.getIndex() != index){
            current = current.next;
            if (current == null) {
                System.out.println("Index not found in LinkedList");
                return false;
            }
        }

        if(current == headLink){
            deleteHeadLink();
        } else if (current == newestLink){
            deleteNewestLink();
        } else {
            //sets the current's previous Link and next Link referencing each
other.
            current.next.previous = current.previous;
            current.previous.next = current.next;
        }

        return true;
    }

    /**
     * This method will print from the first index added and check how many
per
     * line to display by passing in an int value.
     *
     * @param perLine number of displays in a row
     */
    public void printLinkedListIndex(int perLine) {

         //starts from the headLink
        Link theLink = headLink;

        while(theLink != null) {

            //prints index in each Link as it passes
            theLink.printLinkIndex();

            Integer number = theLink.getIndex() % perLine;
            if (number.equals(0)) {
                System.out.println();
            }

            //sets the link to the previous Link
            theLink = theLink.next;

        }
    }

    /**
    * This method will print the characters from beginning and check how many
per
    * line to display by passing in an int value.
    *
    * @param perLine number of displays in a row
```

```
    */
    public void printLinkedListColor(int perLine) {

         //starts from the headLink
        Link theLink = headLink;

        while(theLink != null) {

            //prints index in each Link as it passes
            theLink.printLinkColor();

            Integer number = theLink.getIndex() % perLine;
            if (number.equals(0)) {
                System.out.println();
            }

            //sets the link to the previous Link
            theLink = theLink.next;

        }
    }

    /**
     * This method will print the DoublyLinkedList in order of a board
rotated
     * to the right
     *
     * @param perLine number of displays in a row
     */
    public void printLinkedListCol(int perLine) {

         //starts from the headLink
        Link theLink = headLink;

        int counter = 1;
        while(theLink != null) {

            //prints index in each Link as it passes
            theLink.printLinkColor();

            if (counter % 6 == 0) {
                System.out.println();
            }

            //sets the link to the previous Link
            theLink = theLink.next;
            counter++;
        }
    }

    /**
     * This method will print from the most recently added index
     */
    public void printLinkedListInReverseOrder() {

        //starts from the newestLink added to the Link List
        Link theLink = newestLink;
```

```
        while(theLink != null) {
            //prints index in each Link as it passes
            theLink.printLinkIndex();

            //sets the link to the previous Link
            theLink = theLink.previous;

            System.out.println();
        }
    }

}

/*
 * File: Project 1
 * Programmer: Jasmine Anica
 * Class: CSC 18C
 * Date: 5/20/15
 */
package project1;


/**
 *
 * @author Jasmine
 */
public class Link {
    //reference to the next link
    Link next;
    //reference to the previous link
    Link previous;

    //number in this link
    private Integer index;
    private Character color;



    /**
     * This method sets the index and color to this Link
     *
     * @param index Integer of Link
     * @param color Character of Link
     */
    public void addLink(Integer index, Character color) {
        this.index = index;
        this.color = color;
    }

    /**
     * This method returns the Link's index Integer
     *
     * @return Integer
     */
    public Integer getIndex(){
        return index;
```

```
    }

    /**
     * This method returns the Link's color Character
     *
     * @return Character
     */
    public Character getCharacter(){
        return color;
    }

    /**
     * This method sets the Link's color Character
     *
     * @param color Character of Link
     */
    public void setCharacter(Character color){
        this.color = color;
    }

    /**
     * This method displays the index of this Link
     */
    public void printLinkIndex() {
        System.out.print(index + " ");
    }

    /**
     * This method displays the color of this Link
     */
    public void printLinkColor(){
        System.out.print(color + " ");
    }
}

/*
 * File: Project 1
 * Programmer: Jasmine Anica
 * Class: CSC 18C
 * Date: 5/20/15
 */
package project1;

/**
 *
 * @author Jasmine
 */
public class Vector {

    private int[] dynamicArray;
    private int size;
    private int index = -1;

    //Constructor
    Vector(){
        //size of array
        size = 12;
```

```
        //initialize the size of the dynamic array
        dynamicArray = new int[size];

        //initialize the elements in the array to -1
        for(int i = 0; i < dynamicArray.length; i++){
            dynamicArray[i] = -1;
        }
    }

    /**
     * This method returns true if array is empty or false if it is filled
     *
     * @return boolean
     */
    public boolean isEmpty() {
        return (index == -1);
    }

    /**
     * This method sets the size of this Vector
     *
     * @param size of Vector
     */
    public void setArraySize(int size) {

        //size of array
        this.size = size;

        //initialize the size of the dynamic array
        dynamicArray = new int[size];

        //initialize the elements in the array to -1
        for(int i = 0; i < dynamicArray.length; i++){
            dynamicArray[i] = -1;
        }
    }

    /**
     * This method adds a number to the Vector and will double size if needed
     *
     * @param number to add to the Vector
     */
    public void add(int number) {

        //check to see if there is room to add
        if (checkAdd()) {
            index++;
            dynamicArray[index] = number; //add number to array

        } else {

            int[] temp = new int[size];
            temp = dynamicArray;

            // doubles the size of array
            size *= 2;
```

```
            //resizes array
            dynamicArray = new int[size];

            //initialize the elements in the array to -1
            for(int i = 0; i < dynamicArray.length; i++){
                dynamicArray[i] = -1;
            }
            //set previous array values to new array
            for (int i = 0; i < temp.length; i++) {
                dynamicArray[i] = temp[i];
            }

            index++;
            dynamicArray[index] = number; //adds number to array
        }
    }

    /**
     * This method checks to make sure the index does not go out of bounds.
     * It will return true if not out of bounds, it will return false if
index
     * is out of bounds.
     *
     * @return boolean
     */
    public boolean checkAdd() {
        return index != dynamicArray.length-1;
    }

    /**
     * This method will sort the array in the Vector and return an int array.
     *
     * @return int[]
     */
    public int[] sortArray(){
        int temp;
        int[] sorted;

        sorted = dynamicArray;

        int loopBound = sorted.length;

        //top refers the the top number of the array as it loops
        for (int top = 0; top < loopBound; top++) {

            //next refers to the numbers after the top in this loop
            for(int next = top + 1; next < loopBound; next++) {

                if(sorted[top] != -1 && sorted[next] != -1) {

                    if (sorted[top] > sorted[next]) {

                        //temp stores the smaller value
                        temp = sorted[next];

                        //swaps the smaller number to the bigger number
```

```
                        sorted[next] = sorted[top];

                        //swaps from bigger number to smaller number
                        sorted[top] = temp;
                    }
                }
            }
        }

        return sorted;
    }

    /**
     * This function sets the passed in array to this Vector's array
     *
     * @param arr - an integer array
     */
    public void setArray(int[] arr){
        this.dynamicArray = arr;
    }


    /**
     * This method will check for any multiple numbers in the Vector and
return
     * true if there is not a duplicate or return false if there is a
duplicate
     *
     * @param num value to check in the array
     * @return boolean
     */
    public boolean checkNoMuliple(int num){

        //searches the array to find a duplicate
        for (int i = 0; i < dynamicArray.length; i++){

            //there already exists a num value in the array
            if(num == dynamicArray[i]){
               return false;
            }
        }

        return true;
    }

    /**
     * This method will return the number at a specific index in the Vector.
     *
     * @param index to a number in the Vector
     * @return int - number in the Vector at passed in index
     */
    public int getNumber(int index){
        return dynamicArray[index];
    }

    /**
     * This method will empty the Vector
```

```
     */
    public void emptyVector(){

        //initialize the elements in the array to -1
        for(int i = 0; i < dynamicArray.length; i++){
            dynamicArray[i] = -1;
        }

        index = -1;
    }

    /**
     * This method will return the size of this Vector
     *
     * @return int - size of Vector
     */
    public int getSize(){
        return (index + 1);
    }


    /**
     * This method will print out the values in the Vector
     *
     * @param perLine number to display in a row
     */
    public void print(int perLine){
        for (int i = 0; i < dynamicArray.length; i++) {
            if(dynamicArray[i] != -1) {
                if(dynamicArray[i] < 10) {
                    System.out.print(dynamicArray[i] + "    ");

                } else {
                    System.out.print(dynamicArray[i] + "   ");
                }
                if(i % perLine == (perLine - 1)) {
                    System.out.println();
                }
            }
        }
    }
}


/*
 * File: Project 1
 * Programmer: Jasmine Anica
 * Class: CSC 18C
 * Date: 5/20/15
 */
package project1;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import static java.lang.Math.pow;
import java.util.HashMap;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Jasmine
 */
public class Levels {

    private HashMap<Integer, String> easyLevels = new
HashMap<Integer,String>();
    private HashMap<Integer, String> hardLevels = new
HashMap<Integer,String>();

    private Vector easyLevelsIndex = new Vector();
    private Vector hardLevelsIndex = new Vector();

    /**
     * Constructor: loads the levels
     */
    Levels(){

        try {
            loadEasyLevels();
        } catch (IOException ex) {
            Logger.getLogger(Levels.class.getName()).log(Level.SEVERE, null,
ex);
        }

        try {
            loadHardLevels();
        } catch (IOException ex) {
            Logger.getLogger(Levels.class.getName()).log(Level.SEVERE, null,
ex);
        }


    }

    /**
     * This method will open the Easy_Levels text file and store the levels
as
     * strings to a Hashmap. The keys are hashed and stored to a Vector.
     * @throws FileNotFoundException
     * @throws IOException
     */
    private void loadEasyLevels() throws FileNotFoundException, IOException{

        // Open the file
        FileInputStream fstream = new FileInputStream("Easy_Levels.txt");
        BufferedReader br = new BufferedReader(new
InputStreamReader(fstream));

        String strLine;
```

```
        //Read File Line By Line
        while ((strLine = br.readLine()) != null)   {
            //hashes the make a key
            Integer value = hashing(strLine);

            //addes key to map with the string
            easyLevels.put(value, strLine);
            //addes key to vector
            easyLevelsIndex.add(value);
        }


        //Close the input stream
        br.close();
    }

    /**
     * This method will open the Hard_Levels text file and store the levels
as
     * strings to a Hashmap. The keys are hashed and stored to a Vector.
     *
     * @throws FileNotFoundException
     * @throws IOException
     */
    private void loadHardLevels() throws FileNotFoundException, IOException{

        // Open the file
        FileInputStream fstream = new FileInputStream("Hard_Levels.txt");
        BufferedReader br = new BufferedReader(new
InputStreamReader(fstream));

        String strLine;

        //Read File Line By Line
        while ((strLine = br.readLine()) != null)   {
            //hashes the make a key
            Integer value = hashing(strLine);

            //addes key to map with the string
            hardLevels.put(value, strLine);
            //addes key to vector
            hardLevelsIndex.add(value);
        }


        //Close the input stream
        br.close();
    }

    /**
     * This method is used to create a unique index for the keys in the
Hashmaps
     * that store the levels
     *
     * @param puzzle - a string with the order of colors
     * @return Integer - the key to a Hashmap
```

```
     */
    private Integer hashing(String puzzle){

        double index = 0;

        for (int i = 0; i < puzzle.length(); i++){
            Character c = puzzle.charAt(i);
            index += c.charValue()* pow(5,i);
        }

        while (index > 200){
            index /= 10;
        }

        return (int) index;

    }

    /**
     * This method will return an easy level
     *
     * @return String
     */
    public String getEasyLevel(){

        return easyLevels.get(getEasyLevelIndex());
    }

    /**
     * This method will return an hard level
     *
     * @return String
     */
    public String getHardLevel(){

        return hardLevels.get(getHardLevelIndex());
    }

    /**
     * This method will choose a random index to pick in the Vector and
return
     * the key value
     *
     * @return Integer
     */
    private Integer getEasyLevelIndex(){

        Random rand = new Random();
        int upperBound = easyLevelsIndex.getSize();
        int lowerBound = 0;

        Integer index;
        index = rand.nextInt(upperBound - lowerBound) + lowerBound;

        index = easyLevelsIndex.getNumber(index);

        return index;
```

```
    }

    /**
     * This method will choose a random index to pick in the Vector and
return
     * the key value
     *
     * @return Integer
     */
    private Integer getHardLevelIndex(){

        Random rand = new Random();
        int upperBound = hardLevelsIndex.getSize();
        int lowerBound = 0;

        Integer index;
        index = rand.nextInt(upperBound - lowerBound) + lowerBound;

        index = hardLevelsIndex.getNumber(index);

        return index;
    }

}
```