Name:	Dan Cassidy
Class:	CSCI-C 490, Mobile Application Development
Assignment:	Homework 4 Part 1
Date:	2015-07-18

```
1  /*-------------------------------------------------------------------------------------------------
2   * Author:      Dan Cassidy
3   * Date:        2015-07-18
4   * Assignment:  HW4-1
5   * Source File: TicTacToe.java
6   * Language:    Java
7   * Course:      CSCI-C 490, Android Programming, MoWe 08:00
8  -------------------------------------------------------------------------------------------------*/
9
10 /**
11  * Main class for the Tic-Tac-Toe game. Constructed using the MVC pattern.
12  *
13  * @author Dan Cassidy
14  */
15 public class TicTacToe
16 {
17     /**
18      * Main entry point for the program.
19      *
20      * @param args Command line arguments.
21      */
22     public static void main(String[] args)
23     {
24         int numRows = 0, numColumns = 0, winLength = 0;
25
26         // Parse command line arguments, exiting if they aren't numbers.
27         try
28         {
29             if (args.length > 1)
30             {
31                 numRows = Integer.parseInt(args[0]);
32                 numColumns = Integer.parseInt(args[1]);
33                 if (args.length > 2)
34                     winLength = Integer.parseInt(args[2]);
35             }
36             else if (args.length == 1)
37                 winLength = Integer.parseInt(args[0]);
38         }
39         catch (NumberFormatException ex)
40         {
41             System.out.println("Command line arguments must be numbers.");
42             System.exit(1);
43         }
44
45         GameController control = new GameController(numRows, numColumns, winLength);
46         control.showView();
47     }
48 }
49
```

```
1  /*-------------------------------------------------------------------------------------------------
2   * Author:      Dan Cassidy
3   * Date:        2015-07-18
4   * Assignment:  HW4-1
5   * Source File: GameController.java
6   * Language:    Java
7   * Course:      CSCI-C 490, Android Programming, MoWe 08:00
8  -------------------------------------------------------------------------------------------------*/
9  import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11
12 /**
13  * Controller class for the Tic Tac Toe game. Bridges the model (Game class) and the view (GameView
14  * class) and associates user generated events with model (data) actions.
15  *
16  * @author Dan Cassidy
17  */
18 public class GameController
19 {
20     private GameModel theGame;  // Model.
21     private GameView theView;   // View.
22
23     /**
24      * 3-parameter constructor.
25      *
26      * @param numRows The number of rows the game board will have.
27      * @param numColumns The number of columns the game board will have.
28      * @param winLength The length of the sequence required to win the game.
29      */
30     public GameController(int numRows, int numColumns, int winLength)
31     {
32         // Set up the model and the view.
33         theGame = new GameModel(numRows, numColumns, winLength);
34         theView = new GameView(theGame.getRows(), theGame.getColumns());
35         theView.setWinningConditionsLabelText(theGame.getWinLength() + " in a row wins");
36         theView.setStatusLabelText(theGame.getStatusString());
37
38         // Add listeners to the view.
39         theView.addResetButtonActionListener(new ActionListener()
40         {
41             public void actionPerformed(ActionEvent evt)
42             {
43                 theGame.reset();
44                 theView.reset();
45                 theView.setStatusLabelText(theGame.getStatusString());
46             }
47         });
48         theView.addBoardButtonActionListener(new ActionListener()
49         {
50             public void actionPerformed(ActionEvent event)
51             {
52                 int row = Integer.parseInt(event.getActionCommand()) / theGame.getColumns();
53                 int column = Integer.parseInt(event.getActionCommand()) % theGame.getColumns();
54
55                 theGame.playMove(row, column);
56                 theView.setBoardButtonText(row, column, theGame.getSpaceString(row, column));
57                 theView.setStatusLabelText(theGame.getStatusString());
58                 if (theGame.getStatus() != GameModel.Status.IN_PROGRESS)
59                     theView.setBoardEnabled(false);
60             }
```

```
61          });
62      }
63
64      /**
65       * Show the view.
66       */
67      public void showView()
68      {
69          theView.setVisible(true);
70      }
71  }
72
```

```
/*-------------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-07-18
 * Assignment:  HW4-1
 * Source File: Game.java
 * Language:    Java
 * Course:      CSCI-C 490, Android Programming, MoWe 08:00
-------------------------------------------------------------------------------------------------*/

/**
 * Model for the Tic-Tac-Toe game.
 *
 * Can scale to an arbitrary board size and use an arbitrary winning sequence length.
 *
 * @author Dan Cassidy
 */
public class GameModel
{
    public static enum Mark { X, O }
    public static enum Status { IN_PROGRESS, X_WIN, O_WIN, DRAW }

    private static final int DEFAULT_NUM_ROWS = 3;
    private static final int DEFAULT_NUM_COLUMNS = 3;
    private static final int DEFAULT_WIN_LENGTH = 3;

    private final int NUM_ROWS;
    private final int NUM_COLUMNS;
    private final int WIN_LENGTH;
    private final int MAX_SPACES;

    private Mark[][] board;
    private Mark turn;
    private Status status;
    private int usedSpaces;

    /**
     * Default constructor. Simply calls the 3-parameter constructor with the default values.
     */
    public GameModel()
    {
        this(DEFAULT_NUM_ROWS, DEFAULT_NUM_COLUMNS, DEFAULT_WIN_LENGTH);
    }

    /**
     * 3-parameter constructor. If there is a problem with an argument, the default value is used.
     *
     * @param rows The number of rows on the game board. Should be >= 3.
     * @param columns The number of columns on the game board. Should be >= 3.
     * @param winLength The length of the sequence required to win. Should be >= 3 and <= the
     * smaller of the number of rows and the number of columns.
     */
    public GameModel(int rows, int columns, int winLength)
    {
        NUM_ROWS = (rows < DEFAULT_NUM_ROWS ? DEFAULT_NUM_ROWS : rows);
        NUM_COLUMNS = (columns < DEFAULT_NUM_COLUMNS ? DEFAULT_NUM_COLUMNS : columns);
        MAX_SPACES = NUM_ROWS * NUM_COLUMNS;
        if (    winLength < DEFAULT_WIN_LENGTH ||
                winLength > (NUM_ROWS > NUM_COLUMNS ? NUM_COLUMNS : NUM_ROWS))
            WIN_LENGTH = DEFAULT_WIN_LENGTH;
        else
```

```
61              WIN_LENGTH = winLength;
62          reset();
63      }
64
65      // BEGIN GETTERS AND SETTERS -->
66      public int getColumns()
67      {
68          return NUM_COLUMNS;
69      }
70
71      public int getRows()
72      {
73          return NUM_ROWS;
74      }
75
76      public String getSpaceString(int row, int column)
77      {
78          if (!validCoords(row, column) || board[row][column] == null)
79              return "";
80          else
81              return board[row][column].toString();
82      }
83
84      public Status getStatus()
85      {
86          return status;
87      }
88
89      public String getStatusString()
90      {
91          switch (status)
92          {
93              case IN_PROGRESS:
94                  return turn + "'s Turn";
95              case X_WIN:
96                  return "X Wins";
97              case O_WIN:
98                  return "O Wins";
99              case DRAW:
100                 return "Draw";
101             default:
102                 return "Unknown Status";
103         }
104     }
105
106     public Mark getTurn()
107     {
108         return turn;
109     }
110
111     public int getWinLength()
112     {
113         return WIN_LENGTH;
114     }
115     // <-- END GETTERS AND SETTERS
116
117     /**
118      * Play a single move at the given game board coordinates.
119      *
120      * @param row The row where the mark should be placed.
```

```
121         * @param column The column where the mark should be placed.
122         */
123        public void playMove(int row, int column)
124        {
125            // If the game had ended, no more moves are accepted.
126            if (status != Status.IN_PROGRESS)
127                return;
128
129            // Verify the row and column values.
130            if (!validCoords(row, column))
131                return;
132
133            // Verify that the destination is empty.
134            if (board[row][column] == null)
135            {
136                usedSpaces++;
137                board[row][column] = turn;
138
139                // Can't be a winning move until at least (WIN_LENGTH * 2 - 1) spaces have been used.
140                if (usedSpaces >= WIN_LENGTH * 2 - 1)
141                    checkBoard();
142
143                turn = (turn == Mark.X ? Mark.O : Mark.X);
144            }
145        }
146
147        /**
148         * Discards the old game board and creates a new one in its place and sets the turn to X, the
149         * game status to in progress, and the number of used spaces to 0.
150         */
151        public void reset()
152        {
153            board = new Mark[NUM_ROWS][NUM_COLUMNS];
154            turn = Mark.X;
155            status = Status.IN_PROGRESS;
156            usedSpaces = 0;
157        }
158
159        /**
160         * Checks the game board to see if there is a winner or a draw.
161         */
162        private void checkBoard()
163        {
164            // Check for winning sequences.
165            if (checkWin())
166                status = (turn == Mark.X ? Status.X_WIN : Status.O_WIN);
167            // Check for a draw.
168            else if (usedSpaces == MAX_SPACES)
169                status = Status.DRAW;
170        }
171
172        /**
173         * Check for a winning sequence recursively in a given 'direction'. Upon first entry into the
174         * method (<b>numSequential</b> = 1), this method does several things to avoid unnecessary
175         * recursions so it can scale well to an arbitrary board size and winning sequence length.
176         * <ul><li>It verifies that the final row/column aren't going to be outside the bounds of the
177         * board.</li>
178         * <li>It checks the neighboring space in the direction of travel to make sure it matches.</li>
179         * <li>It checks the final destination space (that is, the space that this method will look at
180         * if it reaches the WIN_LENGTH'th depth) to make sure it matches.</li></ul>
```

```
181         *
182         * @param row The row portion of the board space being looked at.
183         * @param column The column portion of the board space being looked at.
184         * @param rowStepOffset The row offset applied each step.
185         * @param columnStepOffset The column offset applied each step.
186         * @param numSequential The number of sequential marks found thus far.
187         * @return boolean, indicating whether a winning sequence has been found (true) or not (false).
188         */
189        private boolean checkSequence(int row, int column, int rowStepOffset, int columnStepOffset,
190                int numSequential)
191        {
192            // Perform initial checks. These are to cut down on the recursion that needs to happen.
193            if (numSequential == 1)
194            {
195                int finalRow = row + rowStepOffset * (WIN_LENGTH - 1);
196                int finalColumn = column + columnStepOffset * (WIN_LENGTH - 1);
197
198                // Bounds check.
199                if (!validCoords(finalRow, finalColumn))
200                    return false;
201
202                // Neighbor check.
203                if (board[row + rowStepOffset][column + columnStepOffset] != turn)
204                    return false;
205
206                // Destination check.
207                if (board[finalRow][finalColumn] != turn)
208                    return false;
209            }
210
211            // Verify that the sequence continues to match.
212            if (board[row][column] != turn)
213                return false;
214            // Check to see if the sequence is of winning length.
215            else if (numSequential == WIN_LENGTH)
216                return true;
217
218            // Move to the next spot in the sequence.
219            return checkSequence(row + rowStepOffset, column + columnStepOffset, rowStepOffset,
220                    columnStepOffset, numSequential + 1);
221        }
222
223        /**
224         * Checks for a winning sequence on the game board. Wrapper for the recursive checkSequence
225         * method.
226         *
227         * @return boolean, indicating whether a winning sequence was found (true) or not (false).
228         */
229        private boolean checkWin()
230        {
231            boolean win = false;
232
233            for (int row = 0; !win && row < NUM_ROWS; row++)
234                for (int column = 0; !win && column < NUM_COLUMNS; column++)
235                    // Only need to check for a winning condition if the board space contains a mark
236                    // that is the same as the current turn. E.g. - Only check for a winning condition
237                    // if it is O's turn and the board contains an 'O' in the current space.
238                    if (board[row][column] == turn)
239                        win =   checkSequence(row, column, 0, 1, 1) ||   // Right.
240                                checkSequence(row, column, 1, 0, 1) ||   // Down.
```

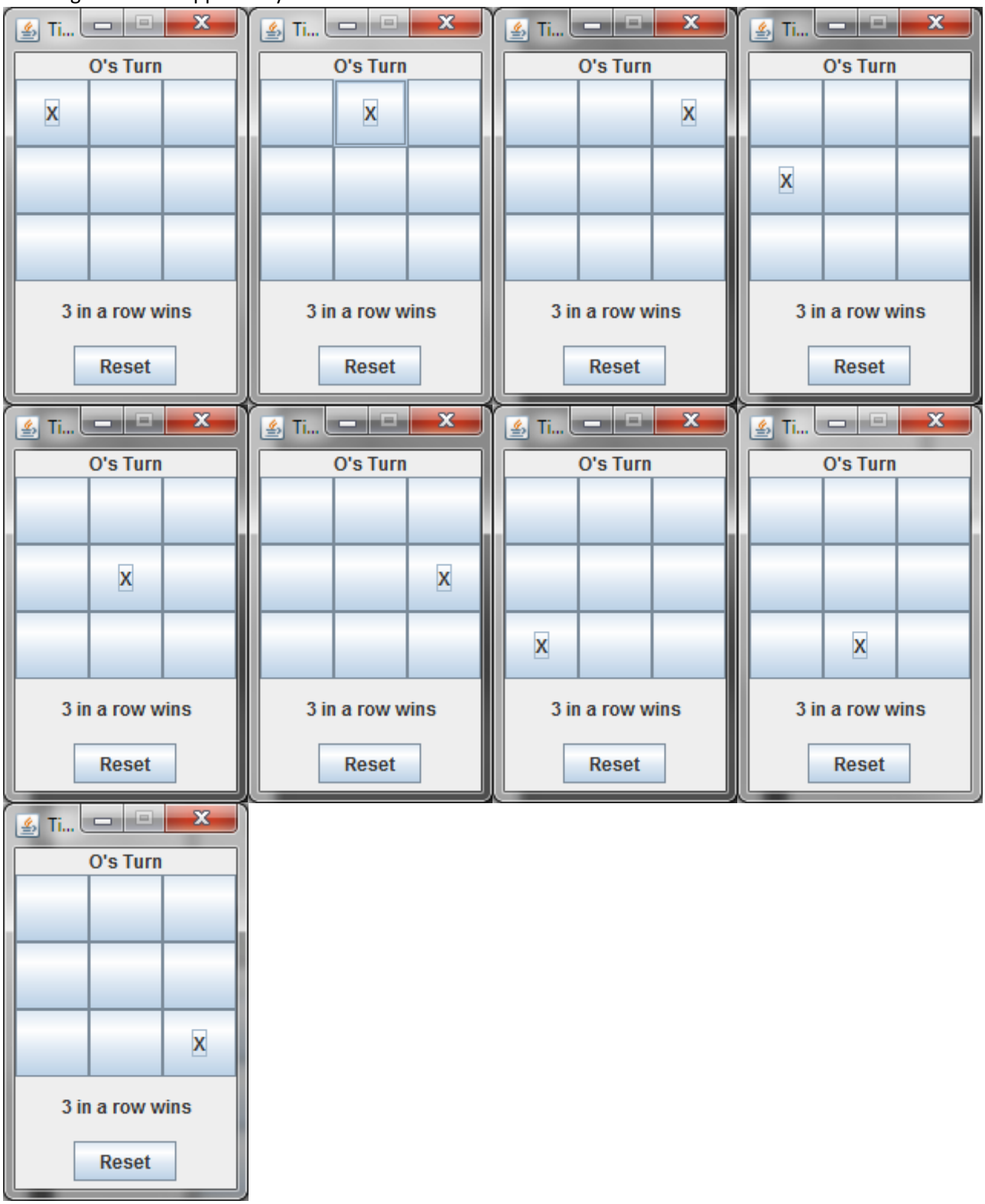
```
241                             checkSequence(row, column, 1, 1, 1) ||   // Diagonal down right.
242                             checkSequence(row, column, -1, 1, 1);    // Diagonal up right.
243
244         return win;
245     }
246
247     /**
248      * Checks the given row and column values to make sure they are valid (within bounds) for the
249      * current game board.
250      *
251      * @param row The row value to check.
252      * @param column The column value to check.
253      * @return boolean, indicating whether the given coordinates are valid (true) or not (false).
254      */
255     private boolean validCoords(int row, int column)
256     {
257         return row >= 0 && row < NUM_ROWS && column >= 0 && column < NUM_COLUMNS;
258     }
259 }
260
```

```
/*-------------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-07-18
 * Assignment:  HW4-1
 * Source File: GameView.java
 * Language:    Java
 * Course:      CSCI-C 490, Android Programming, MoWe 08:00
-------------------------------------------------------------------------------------------------*/
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;

/**
 * View for the Tic-Tac-Toe game. Handles the visual representation.
 *
 * @author Dan Cassidy
 */
@SuppressWarnings("serial")
public class GameView extends JFrame
{
    private JLabel statusLabel;
    private JButton[][] board;
    private JLabel winningConditionsLabel;
    private JButton resetButton = new JButton("Reset");

    /**
     * 2-parameter constructor.
     *
     * @param numRows The number of rows of buttons the game board will have.
     * @param numColumns The number of columns of buttons the game board will have.
     */
    public GameView(int numRows, int numColumns)
    {
        // General window options and layout.
        super("Tic Tac Toe");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setResizable(false);
        setSize(numColumns * 50, (numRows + 2) * 50);
        setLayout(new BorderLayout());

        // NORTH. Create and add the game status label.
        statusLabel = new JLabel();
        statusLabel.setHorizontalAlignment(SwingConstants.CENTER);
        add(statusLabel, BorderLayout.NORTH);

        // CENTER. Create and add the buttons for the board.
        JPanel boardPanel = new JPanel(new GridLayout(numRows, numColumns));
        board = new JButton[numRows][numColumns];
        for (int row = 0; row < numRows; row++)
            for (int column = 0; column < numColumns; column++)
            {
                board[row][column] = new JButton();
                board[row][column].setActionCommand("" + (row * numColumns + column));
```

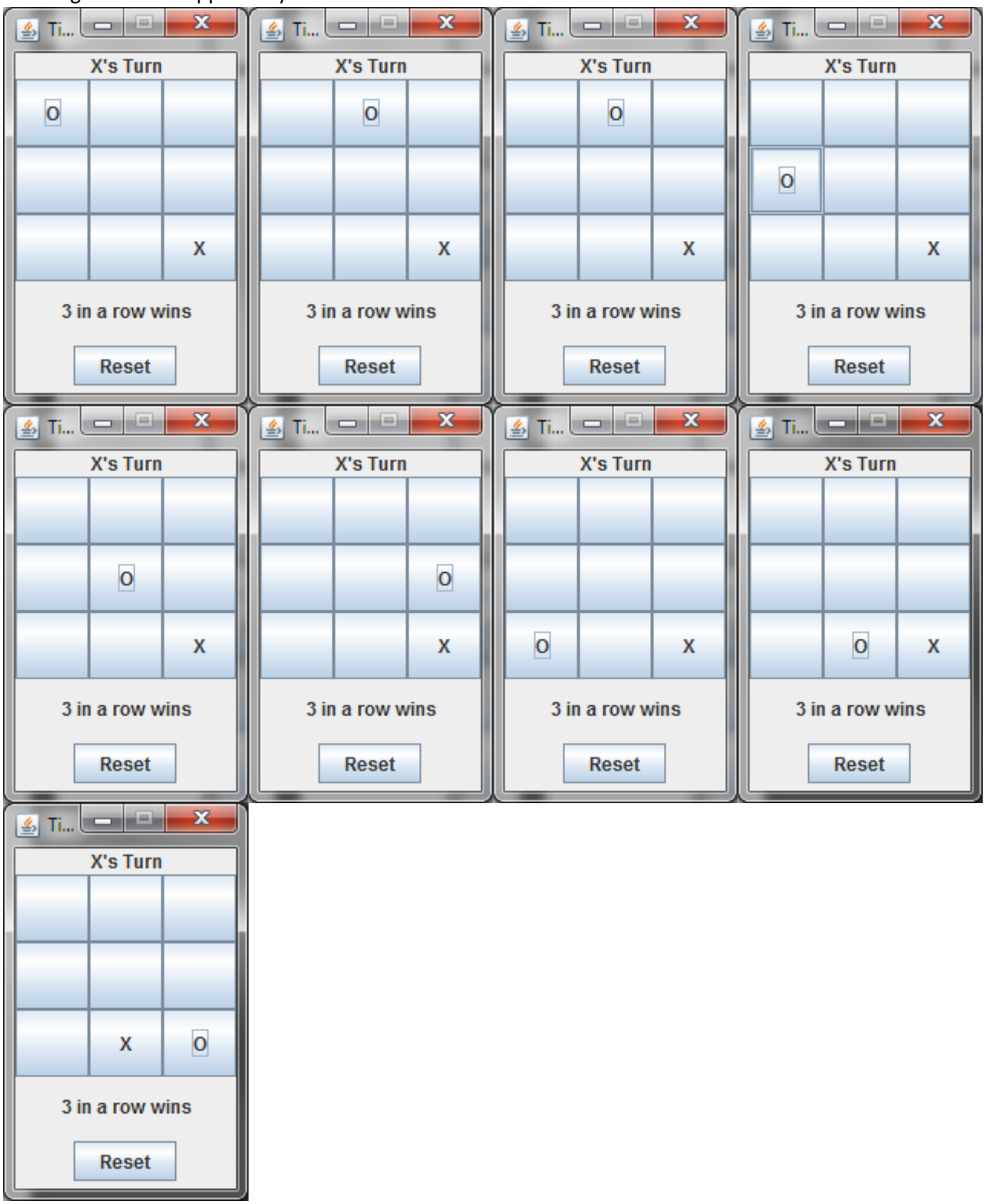
```
61                  boardPanel.add(board[row][column]);
62              }
63          add(boardPanel, BorderLayout.CENTER);
64
65          // SOUTH. Create and add a label detailing the winning conditions and a reset button.
66          winningConditionsLabel = new JLabel();
67          winningConditionsLabel.setHorizontalAlignment(SwingConstants.CENTER);
68          JPanel resetButtonPanel = new JPanel();
69          resetButtonPanel.setLayout(new FlowLayout());
70          resetButtonPanel.add(resetButton);
71          JPanel bottomPanel = new JPanel(new GridLayout(2,1));
72          bottomPanel.add(winningConditionsLabel);
73          bottomPanel.add(resetButtonPanel);
74          add(bottomPanel, BorderLayout.SOUTH);
75      }
76
77      /**
78       * Add an action listener to all of the board buttons.
79       *
80       * @param listener The ActionListener to add to the buttons.
81       */
82      public void addBoardButtonActionListener(ActionListener listener)
83      {
84          for (JButton[] buttonRow : board)
85              for (JButton button : buttonRow)
86                  button.addActionListener(listener);
87      }
88
89      /**
90       * Add an action listener to the reset button.
91       *
92       * @param listener The ActionListener to add to the reset button.
93       */
94      public void addResetButtonActionListener(ActionListener listener)
95      {
96          resetButton.addActionListener(listener);
97      }
98
99      /**
100      * Resets the status text and the board buttons to default.
101      */
102     public void reset()
103     {
104         statusLabel.setText("");
105         setBoardEnabled(true);
106
107         for (JButton[] buttonRow : board)
108             for (JButton button : buttonRow)
109                 button.setText("");
110     }
111
112     /**
113      * Interface method to set the text of a given board button.
114      *
115      * @param row The board row of the button.
116      * @param column The board column of the button.
117      * @param text The String to set the text to.
118      */
119     public void setBoardButtonText(int row, int column, String text)
120     {
```

```
121             board[row][column].setText(text);
122         }
123
124         /**
125          * Enables (or disables) the button board.
126          *
127          * @param b true to enable the button board, otherwise false.
128          */
129         public void setBoardEnabled(boolean b)
130         {
131             for (JButton[] buttonRow : board)
132                 for (JButton button : buttonRow)
133                     button.setEnabled(b);
134         }
135
136         /**
137          * Interface method to update the status label.
138          *
139          * @param text The String to set the text to.
140          */
141         public void setStatusLabelText(String text)
142         {
143             statusLabel.setText(text);
144         }
145
146         /**
147          * Interface method to update the winning conditions label.
148          *
149          * @param text The String to set the text to.
150          */
151         public void setWinningConditionsLabelText(String text)
152         {
153             winningConditionsLabel.setText(text);
154         }
155     }
156
```

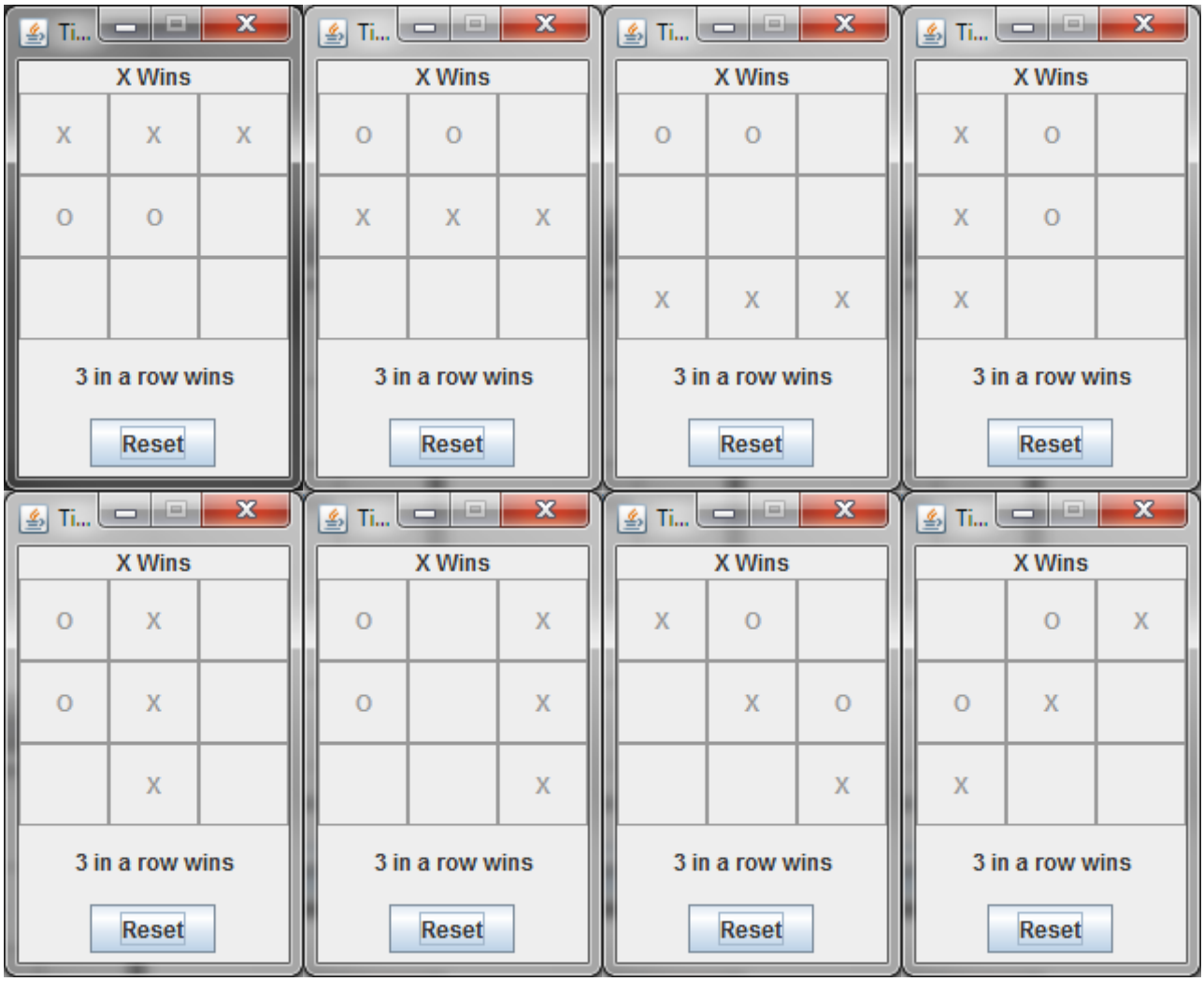
Showing that X can appear anywhere.



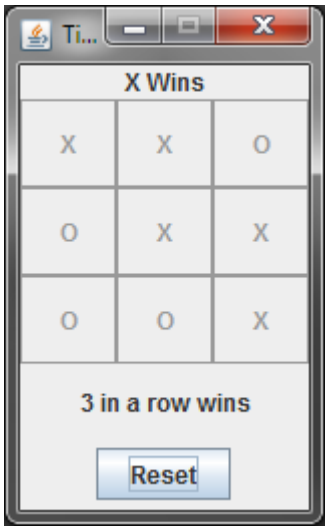
Showing that O can appear anywhere.



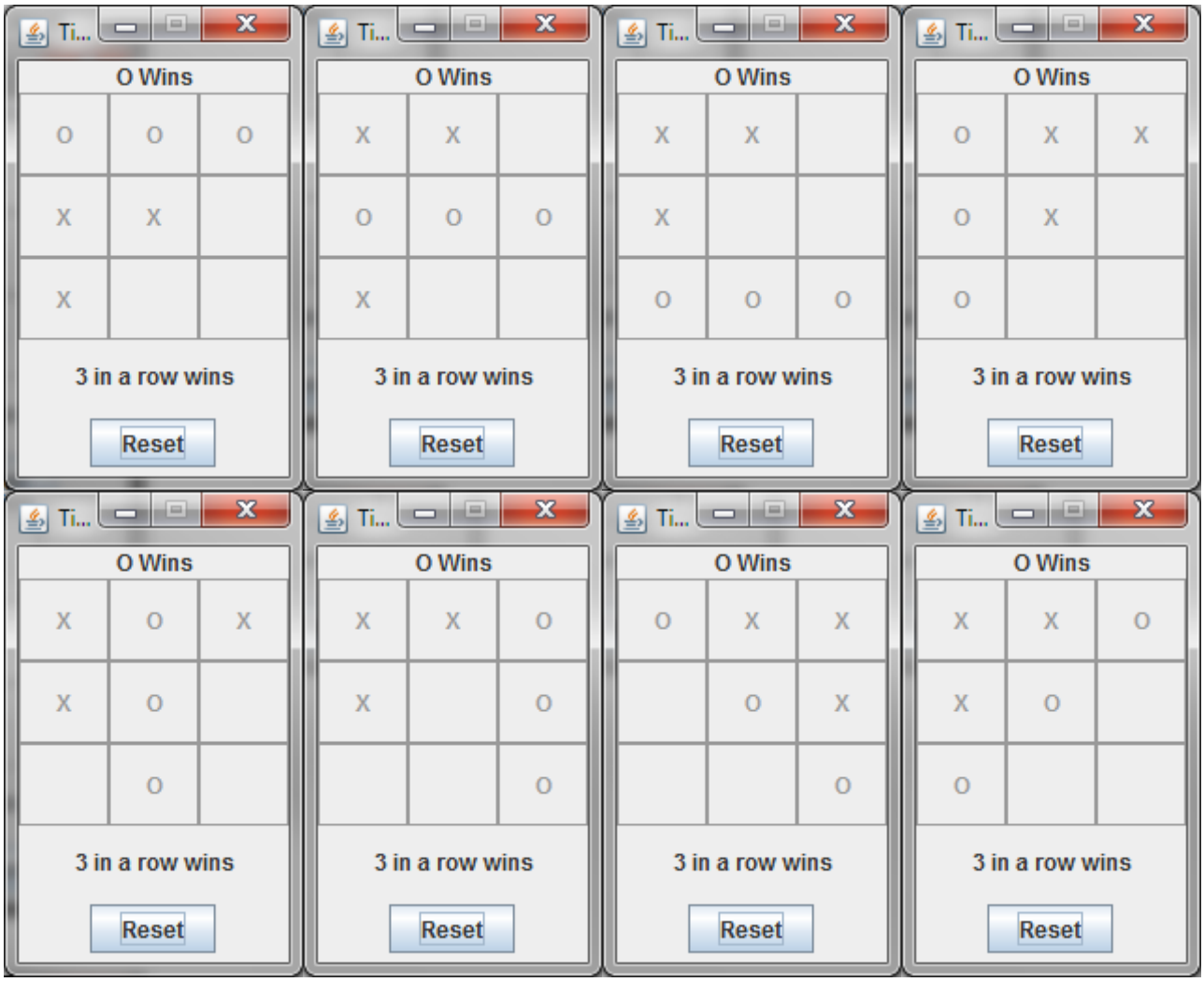
The different X wins.



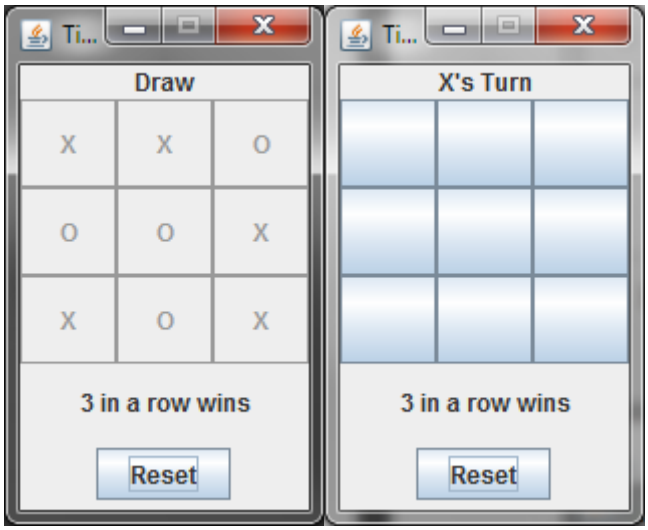
Last move win for X.



The different O wins.



Draw. => Reset.

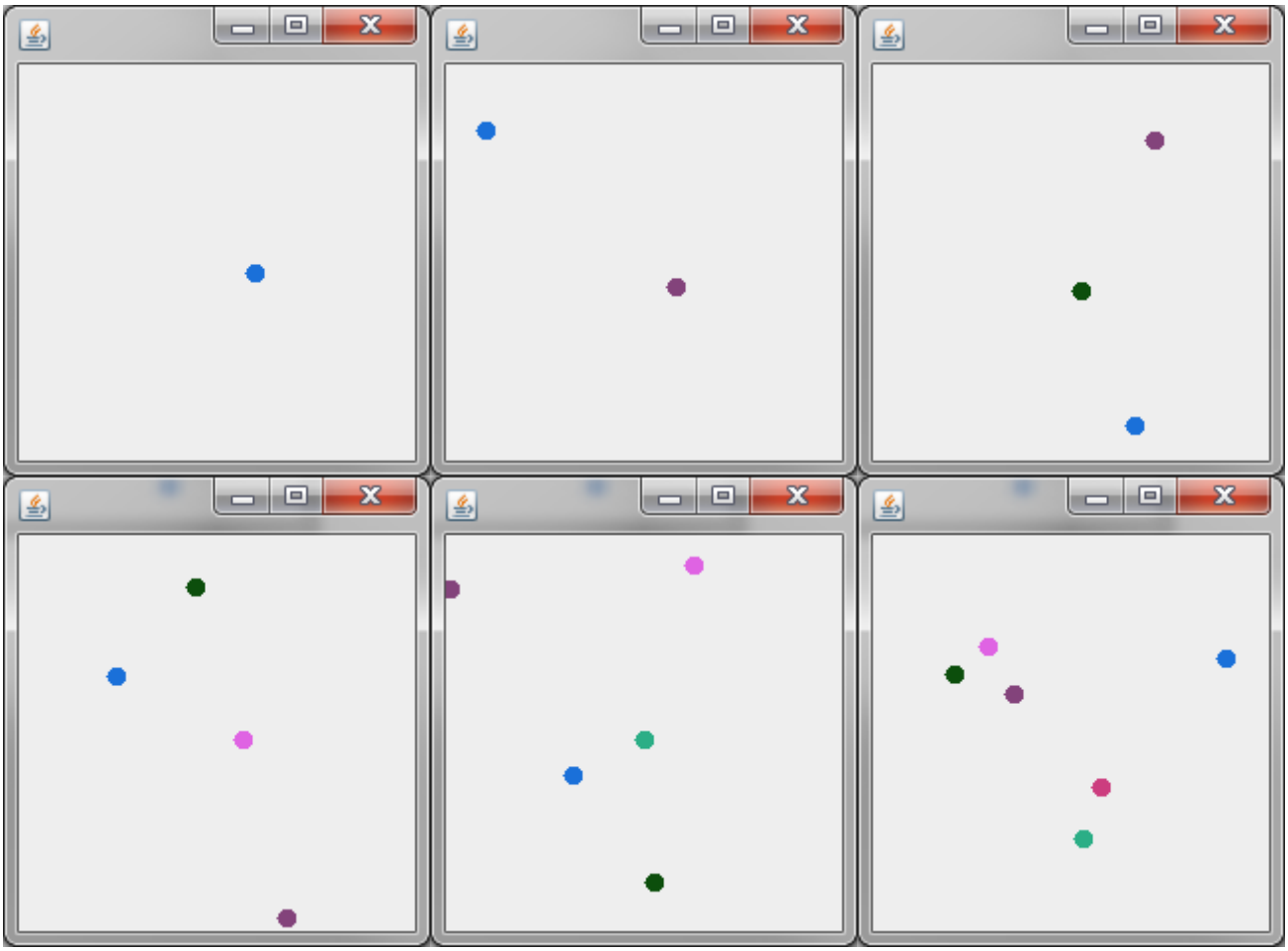


Name:	Dan Cassidy
Class:	CSCI-C 490, Mobile Application Development
Assignment:	Homework 4 Part 2
Date:	2015-07-18

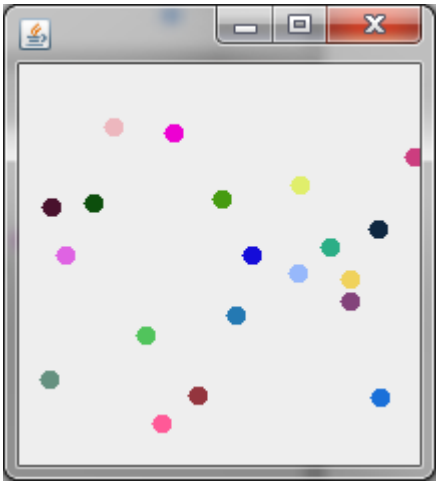
```
1   public class BallPanel extends JPanel
2   {
3       private final int       MAX_BALLS     = 20;                  // maximum number of balls
4       private Ball[]          balls         = new Ball[MAX_BALLS]; // array to hold the balls
5       private int             ballCount     = 0;                   // current number of balls
6
7       // create a ball and set it in motion if no ball exists
8       private void createBall(MouseEvent event)
9       {
10          // if ( blueBall == null ) // if no ball exists
11          if (ballCount < MAX_BALLS)
12          {
13              int x = event.getX(); // get x position of mouse press
14              int y = event.getY(); // get y position of mouse press
15              balls[ballCount] = new Ball(parent, x, y); // create new ball
16              threadExecutor.execute(balls[ballCount++]); // set ball in motion and increment count
17          } // end if
18      } // end method createBall
19
20      // draw ball at current position
21      public void paintComponent(Graphics g)
22      {
23          super.paintComponent(g);
24
25          // Loop through all the existing balls.
26          for (int ballIndex = 0; ballIndex < ballCount; ballIndex++)
27          {
28              g.setColor(balls[ballIndex].getColor()); // Sets color.
29
30              // draw ball
31              g.fillOval(balls[ballIndex].getX(), balls[ballIndex].getY(), 10, 10);
32          } // end if
33      } // end method paintComponent
34  } // end class BallPanel
```

```
1   import java.awt.Color;
2
3   public class Ball implements Runnable
4   {
5       private Color ballColor = new Color(generator.nextInt(256), generator.nextInt(256),
6               generator.nextInt(256));           // make the ball a random color
7
8       // get color of ball
9       public Color getColor()
10      {
11          return ballColor;
12      }
13  } // end class Ball
14
```

1-6 balls.



20 balls.



Name:	Dan Cassidy
Class:	CSCI-C 490, Mobile Application Development
Assignment:	Homework 4 Part 3
Date:	2015-07-20

```
/*-----------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-07-20
 * Assignment:  HW4-1
 * Source File: Program.java
 * Language:    Java
 * Course:      CSCI-C 490, Android Programming, MoWe 08:00
-----------------------------------------------------------------------------------------------*/
import javax.swing.UIManager;

/**
 * Main class for the Blank Trimmer program. Uses a modified MVC pattern as the BlankTrimmer class
 * is just a utility class and doesn't require instantiation.
 *
 * @author Dan Cassidy
 */
public class Program
{
    /**
     * Main entry point for the program.
     *
     * @param args Command line arguments. <i>Not used</i>.
     */
    public static void main(String[] args)
    {
        // Use the system's look and feel if possible.
        try
        {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch (Exception ex)
        {
            System.err.println("Something went wrong trying to set the system look and feel.");
            System.err.println("Using the default.");
        }

        // Create new controller and show the GUI.
        BlankTrimmerController control = new BlankTrimmerController();
        control.showView();
    }
}

```

```
/*-------------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-07-20
 * Assignment:  HW4-3
 * Source File: BlankTrimmerController.java
 * Language:    Java
 * Course:      CSCI-C 490, Android Programming, MoWe 08:00
-------------------------------------------------------------------------------------------------*/
import java.awt.event.*;
import java.io.IOException;
import java.nio.file.*;

import javax.swing.JOptionPane;

/**
 * Controller for the Blank Trimmer program.
 *
 * @author Dan Cassidy
 */
public class BlankTrimmerController
{
    BlankTrimmerView theView = new BlankTrimmerView();

    /**
     * No-parameter constructor.
     */
    public BlankTrimmerController()
    {
        // Add action listener to the trim button in the view.
        theView.addTrimButtonActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                String message = "";
                int messageType = 0;

                try
                {
                    // Try to trim the file.
                    BlankTrimmer.trim(theView.getFilePath());
                    message = "Blanks trimmed successfully.";
                    messageType = JOptionPane.INFORMATION_MESSAGE;
                }
                catch (InvalidPathException ex)
                {
                    // Path couldn't be parsed. Not a valid path or a bad character or something.
                    message = "Invalid file path.\n" + ex.getReason();
                    messageType = JOptionPane.ERROR_MESSAGE;
                }
                catch (NoSuchFileException ex)
                {
                    // File couldn't be found.
                    message = "File does not exist.";
                    messageType = JOptionPane.ERROR_MESSAGE;
                }
                catch (SecurityException ex)
                {
                    // A file couldn't be read or couldn't be written to.
                    message = "Access denied.\n" + ex.getMessage();
                    messageType = JOptionPane.ERROR_MESSAGE;
```

```
61                  }
62                  catch (IOException ex)
63                  {
64                      // Generic I/O exception.
65                      message = "I/O exception:\n" + ex.getMessage();
66                      messageType = JOptionPane.ERROR_MESSAGE;
67                  }
68                  catch (Exception ex)
69                  {
70                      // Some other exception that wasn't explicitly planned for.
71                      message = "Unknown error:\n" + ex.getMessage();
72                      messageType = JOptionPane.ERROR_MESSAGE;
73                      ex.printStackTrace();
74                  }
75                  finally
76                  {
77                      // Always display a message as to how the operation went.
78                      theView.showMessage(message, messageType);
79                  }
80              }
81          });
82      }
83
84      /**
85       * Show the view.
86       */
87      public void showView()
88      {
89          theView.setVisible(true);
90      }
91  }
92
```

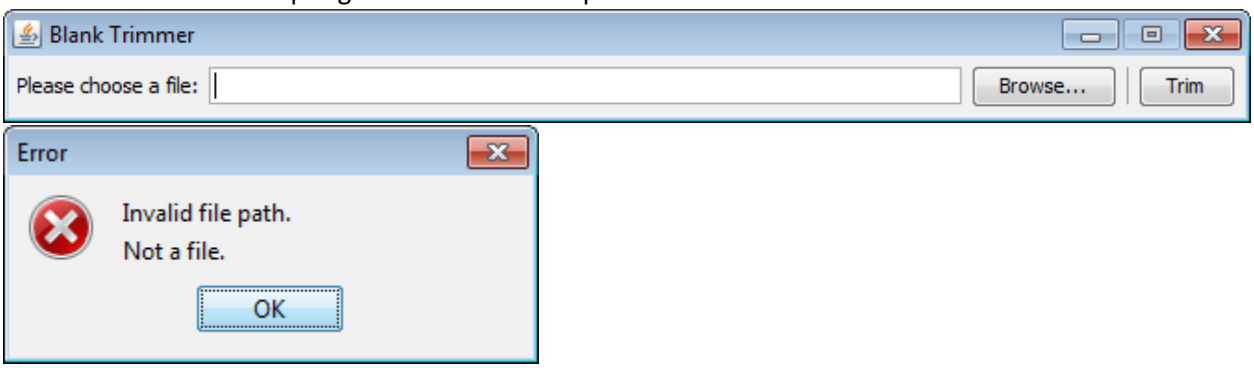
```
1  /*-------------------------------------------------------------------------------------------------------
2   * Author:      Dan Cassidy
3   * Date:        2015-07-20
4   * Assignment:  HW4-3
5   * Source File: BlankTrimmerView.java
6   * Language:    Java
7   * Course:      CSCI-C 490, Android Programming, MoWe 08:00
8   -------------------------------------------------------------------------------------------------------*/
9  import java.awt.*;
10 import java.awt.event.*;
11
12 import javax.swing.*;
13
14 /**
15  * View for the Blank Trimmer program.
16  *
17  * @author Dan Cassidy
18  */
19 @SuppressWarnings("serial")
20 public class BlankTrimmerView extends JFrame
21 {
22     private JTextField filePath = new JTextField();
23     private JButton trimButton = new JButton("Trim");
24
25     /**
26      * No-parameter constructor. Sets up the frame for display.
27      */
28     public BlankTrimmerView()
29     {
30         super("Blank Trimmer");
31         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
32         setResizable(false);
33         setLayout(new FlowLayout());
34
35         // Component 1.
36         add(new JLabel("Please choose a file:"));
37
38         // Component 2.
39         filePath.setColumns(50);
40         add(filePath);
41
42         // Component 3.
43         JButton browseButton = new JButton("Browse...");
44         browseButton.addActionListener(new ActionListener()
45         {
46             // Use the file chooser dialog to get a file name.
47             public void actionPerformed(ActionEvent event)
48             {
49                 final JFileChooser fileChooser = new JFileChooser();
50                 int result = fileChooser.showOpenDialog(BlankTrimmerView.this);
51                 if (result == JFileChooser.APPROVE_OPTION)
52                     filePath.setText(fileChooser.getSelectedFile().getAbsolutePath());
53             }
54         });
55         add(browseButton);
56
57         // Component 4.
58         JSeparator separator = new JSeparator(SwingConstants.VERTICAL);
59         separator.setPreferredSize(new Dimension(1, 20));
60         add(separator);
```

```
61
62          // Component 5.
63          add(trimButton);
64
65          // Force the frame resize itself and make it appear where the host system dictates.
66          pack();
67          setLocationByPlatform(true);
68      }
69
70      /**
71       * Add an action listener to the trim button.
72       *
73       * @param listener The ActionListener to add.
74       */
75      public void addTrimButtonActionListener(ActionListener listener)
76      {
77          trimButton.addActionListener(listener);
78      }
79
80      /**
81       * Interface method to get the path of the file to work with.
82       *
83       * @return String containing the path of the file.
84       */
85      public String getFilePath()
86      {
87          return filePath.getText();
88      }
89
90      /**
91       * Show a modal dialog displaying the given message.
92       *
93       * @param message The message to display in the dialog.
94       * @param messageType JOptionPane constant determining the look and feel of the dialog.
95       */
96      public void showMessage(String message, int messageType)
97      {
98          if (message == null)
99              message = "No message.";
100
101         String title = "Message";
102
103         // Only expecting error messages or informational messages, so only checking for those two
104         // cases.
105         switch (messageType)
106         {
107             case JOptionPane.ERROR_MESSAGE:
108                 title = "Error";
109                 break;
110
111             case JOptionPane.INFORMATION_MESSAGE:
112                 title = "Information";
113                 break;
114         }
115
116         JOptionPane.showMessageDialog(this, message, title, messageType);
117     }
118 }
119
```

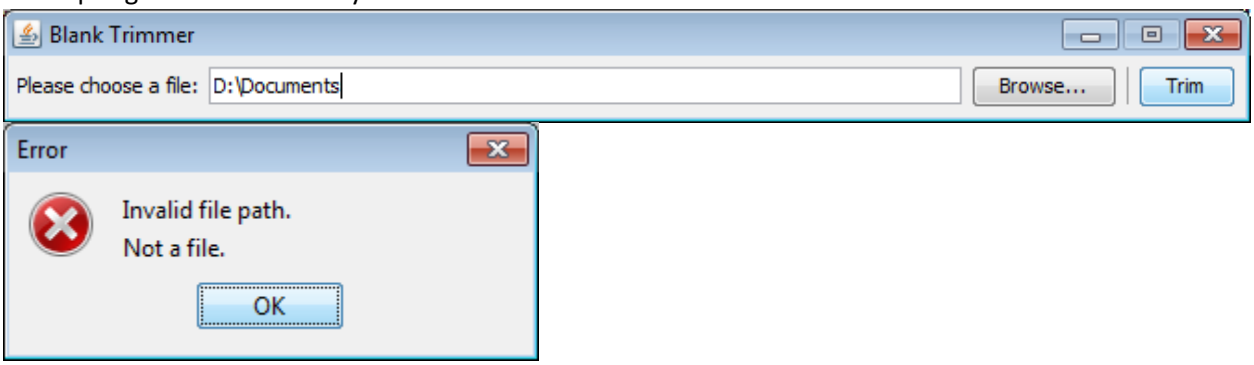
```
1   /*-------------------------------------------------------------------------------------------------
2    * Author:      Dan Cassidy
3    * Date:        2015-07-20
4    * Assignment:  HW4-3
5    * Source File: BlankTrimmer.java
6    * Language:    Java
7    * Course:      CSCI-C 490, Android Programming, MoWe 08:00
8   -------------------------------------------------------------------------------------------------*/
9   import static java.nio.file.StandardCopyOption.*;
10  import static java.nio.file.StandardOpenOption.*;
11
12  import java.io.*;
13  import java.nio.file.*;
14  import java.util.*;
15
16  /**
17   * Utility class to trim extra blanks from a file.
18   *
19   * @author Dan Cassidy
20   */
21  public class BlankTrimmer
22  {
23      /**
24       * Trims any extra blanks from the passed file.
25       *
26       * @param filePath The file to be trimmed.
27       * @throws InvalidPathException if the file path is not a file or the path cannot be parsed.
28       * @throws IOException if there is some other I/O exception.
29       * @throws NoSuchFileException
30       * @throws SecurityException
31       */
32      public static void trim(String filePath) throws IOException, NoSuchFileException
33      {
34          Path mainFile;
35          Path tempFile;
36
37          // Check main file.
38          mainFile = Paths.get(filePath);
39          if (Files.notExists(mainFile))
40              throw new NoSuchFileException(mainFile.toString());
41          if (Files.isDirectory(mainFile))
42              throw new InvalidPathException(mainFile.toString(), "Not a file.");
43          if (!Files.isReadable(mainFile))
44              throw new SecurityException("Main file could not be read.");
45          mainFile = mainFile.toRealPath();
46
47          // Generate and check temp file name.
48          do
49          {
50              tempFile = Paths.get(mainFile.getParent().toString(), createRandomFileName());
51          } while (Files.exists(tempFile));
52
53          // Open main and temp files.
54          try (   InputStream in = Files.newInputStream(mainFile);
55                  OutputStream out = Files.newOutputStream(tempFile, CREATE_NEW))
56          {
57              BufferedReader reader = new BufferedReader(new InputStreamReader(in));
58              BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(out));
59
60              // Copy text from main file to temp file, trimming blanks along the way.
```

```
61              String line = null;
62              while ((line = reader.readLine()) != null)
63              {
64                  StringTokenizer tokenizer = new StringTokenizer(line, " ");
65                  while (tokenizer.hasMoreTokens())
66                  {
67                      writer.append(tokenizer.nextToken());
68                      writer.append((tokenizer.hasMoreTokens() ? " " : "\n"));
69                  }
70              }
71
72              // Force the writer to write everything to the file.
73              writer.flush();
74
75              // Automatically close main and temp files.
76          }
77
78          // Move the temp file to the main file, overwriting in the process.
79          Files.move(tempFile, mainFile, REPLACE_EXISTING);
80      }
81
82      /**
83       * Generates a random file name with the prefix of "Temp" followed by 16 random characters from
84       * A-Z, with a file extension of ".tmp".
85       *
86       * @return String containing the generated file name.
87       */
88      private static String createRandomFileName()
89      {
90          // Define the prefix, suffix, and extension of the file name, as well as how many random
91          // characters are generated.
92          String fileNamePrefix = "Temp";
93          String fileNameSuffix = "";
94          String fileNameExtension = ".tmp";
95          int numRandomChars = 16;
96
97          Random generator = new Random();
98          String randomFileName = "";
99
100         // Generate the random characters.
101         for (int numChars = 1; numChars <= numRandomChars; numChars++)
102             randomFileName += (char)(generator.nextInt(26) + 'A');
103
104         // Return the amalgam.
105         return fileNamePrefix + randomFileName + fileNameSuffix + fileNameExtension;
106     }
107 }
108
```

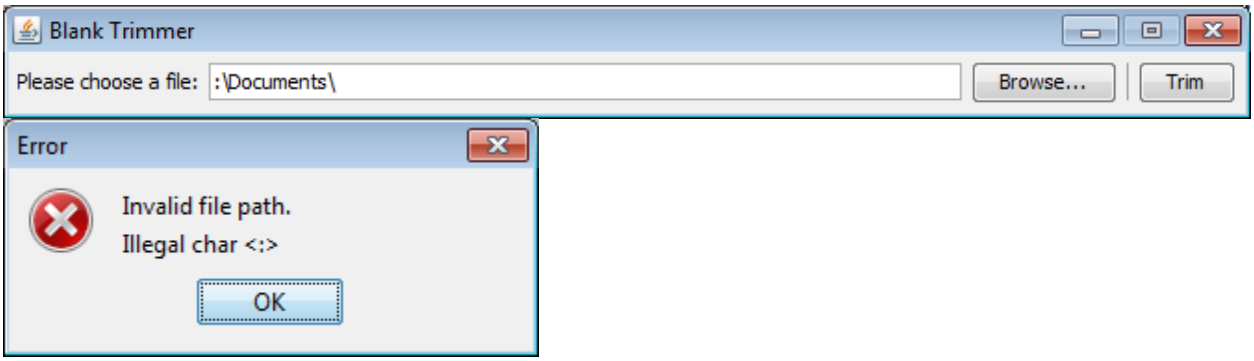
Base window and attempting to trim a blank file path.



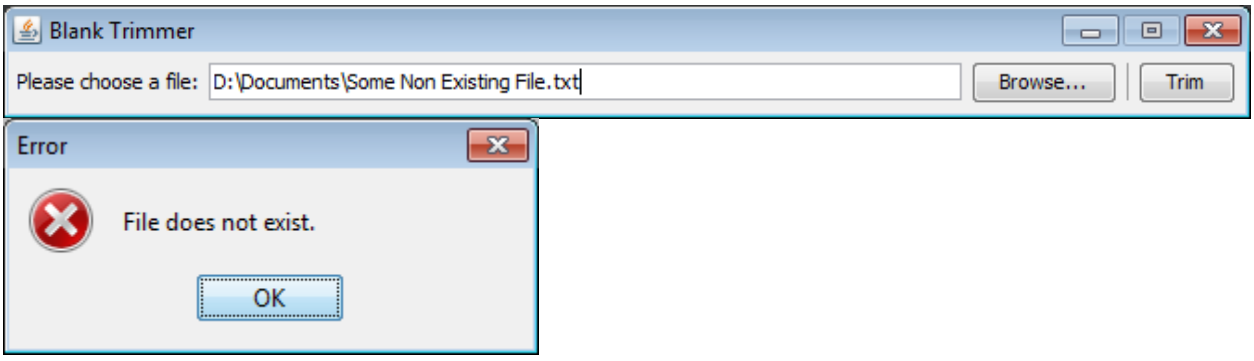
Attempting to trim a directory.



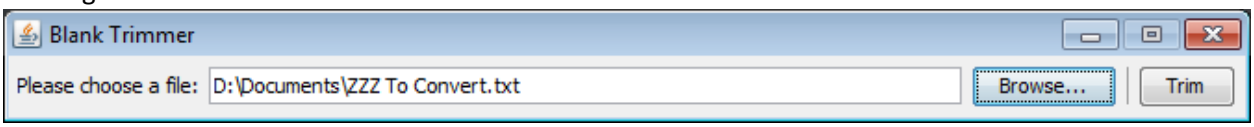
Attempting to trim a bad file path.



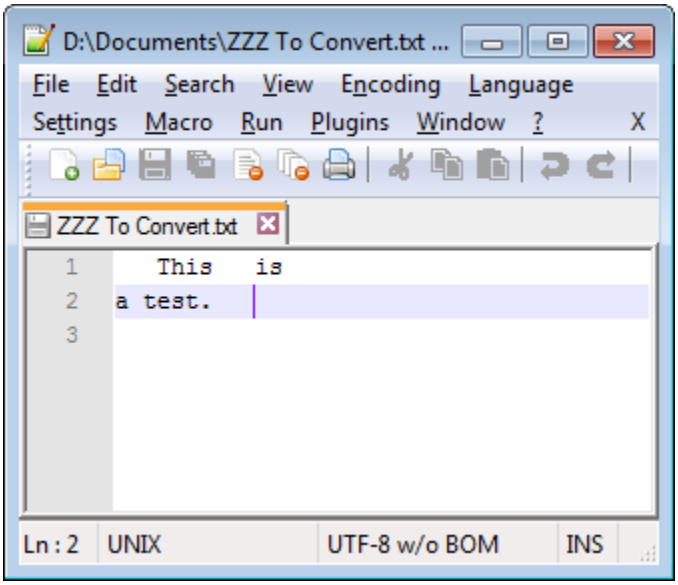
Attempting to trim a non-existent file.



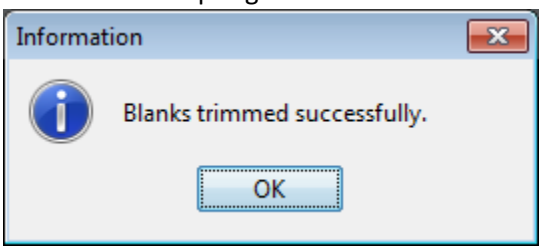
Loading test file to trim.



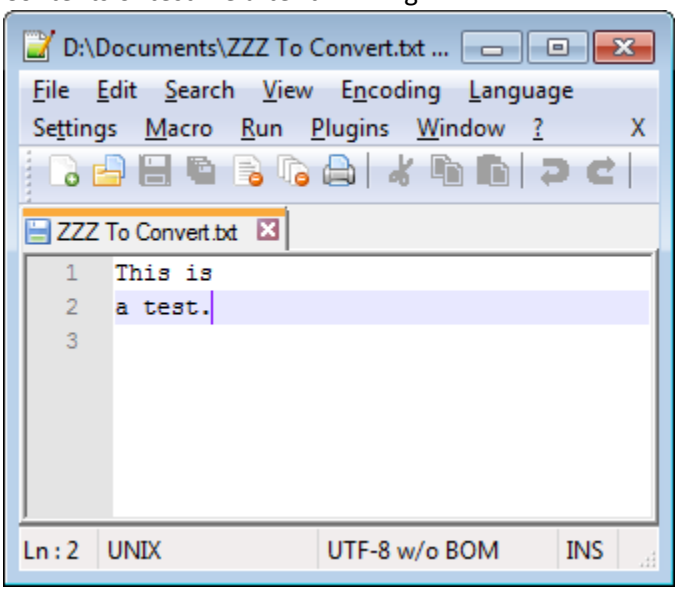
Contents of test file.



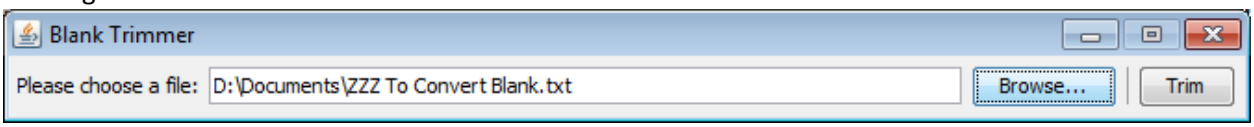
Results of attempting to trim file.



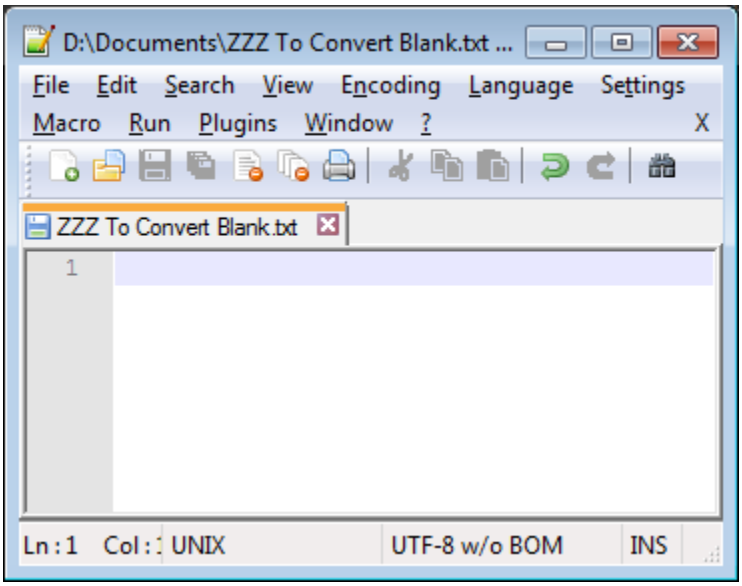
Contents of test file after trimming.



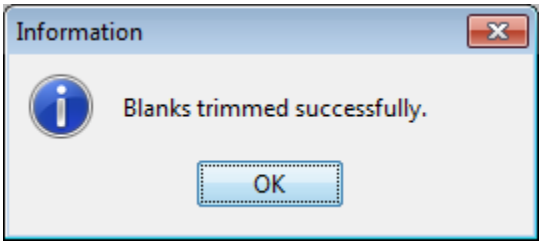
Loading blank test file to trim.



Contents of test file.



Results of attempting to trim file.



Contents of test file after trimming. (No change.)