```java
import java.util.Scanner;

/**
 *
 * @author grantelgin StackDriver methods draw the board, listen for input,
 *         validate input, and save the validated move to a StackManager object
 *
 */

public class StackDriver {
    private StackManager currentStack = new StackManager();
    public Scanner keyboard = new Scanner(System.in);

    public static void main(String[] args) {
        StackDriver me = new StackDriver();
        me.doIt();
    }

    public void doIt() {
        showInstructions();
        showBoard();
        listenForInput();
    }

    public void showInstructions() {
        System.out.println("8 queens v 1.0\n");
        System.out.println("Select a column and a row for each position.\nThe goal is locate 8 queens on the board where they can not kill each other.");
        System.out.println("Enter 999 to exit. ");
    }

    public void listenForInput() {
        // user inputs an int for column, then an int for row. calls to
        // checkLocation validate the input.
        System.out.println("Choose a column: ");
        try {
            int col = keyboard.nextInt();
            if (col == 999){
```

```
38                  System.out.println("Exiting game...");
39                  System.exit(0);
40              }
41              if (col > 0 && col < 9) {
42                  System.out.println("Choose a row: ");
43                  int row = keyboard.nextInt();
44                  if (row == 999) {
45                      System.out.println("Exiting game...");
46                      System.exit(0);
47                  }
48                  if (row > 0 && row < 9) {
49                      System.out.println("column: " + col + "\nrow: " +
  row);
50                      checkLocation(col, row);
51                  } else {
52                      System.out
53                              .println("Woops! Please enter a number
  between 1 and 8");
54                      listenForInput();
55                  }
56              } else {
57                  System.out
58                          .println("Woops! Please enter a number between
  1 and 8");
59                  listenForInput();
60              }
61          } catch (Exception e) {
62              System.out
63                      .println("Woops! Something went wrong. Please
  enter a number between 1 and 8");
64              keyboard.nextLine();
65              listenForInput();
66          }
67      }
68
69      public void checkLocation(int col, int row) {
70          // check the input against current entries in currentStack.
  Any
71          // stackNode that matches col or row returns false.
72          // if valid, check the diagonals and set boolean success. If
  success,
73          // add the node to the stack.
```

```
74          StackNode node = currentStack.getTop();
75          boolean valid = true;
76          for (int x = 0; x < currentStack.getCount(); x++) {
77              if (col == node.getColumn() || row == node.getRow()) {
78                  System.out.println("Nope. Already a queen there");
79                  valid = false;
80                  listenForInput();
81              } else {
82                  node = node.getNext();
83                  valid = true;
84              }
85          }
86          if (valid) {
87              // check diagonal
88              boolean success = (topLeft(col, row) && topRight(col, row)
89                      && bottomLeft(col, row) && bottomRight(col, row));
90
91              if (success) {
92                  // Add to currentStack
93                  System.out.println("success ");
94                  StackNode move = new StackNode();
95                  move.setLocation(col, row);
96                  currentStack.push(move);
97
98                  if (currentStack.getCount() == 8) {
99                      System.out.println("You win!\n");
100                     showBoard();
101                     System.exit(0);
102                 } else
103                     showBoard();
104                     listenForInput();
105             } else {
106                 listenForInput();
107             }
108         }
109
110     }
111
112     public boolean topLeft(int col, int row) {
113         // check for queens on the top left diagonal
114         boolean success = true;
115         StackNode currentNode = currentStack.getTop();
```

```
116         while (col > 0 && row > 0) {
117             for (int x = 0; x < currentStack.getCount(); x++) {
118                 if (col == currentNode.getColumn()
119                         && row == currentNode.getRow()) {
120                     System.out.println("Nope. Queen on top left
    diagonal");
121                     return false;
122                 }
123             }
124 
125             if (success) {
126                 col = col - 1;
127                 row = row - 1;
128             }
129         }
130 
131         return success;
132     }
133 
134     public boolean topRight(int col, int row) {
135         // check for queens on the top right diagonal
136         boolean success = true;
137         StackNode currentNode = currentStack.getTop();
138         while (col > 0 && col < 9 && row > 0 && row < 9) {
139             for (int x = 0; x < currentStack.getCount(); x++) {
140                 if (col == currentNode.getColumn()
141                         && row == currentNode.getRow()) {
142                     System.out.println("Nope. Queen on top right
    diagonal");
143                     return false;
144                 }
145             }
146 
147             if (success) {
148                 col = col + 1;
149                 row = row - 1;
150             }
151         }
152 
153         return success;
154     }
155 
```

```
156     public boolean bottomLeft(int col, int row) {
157         // check for queens on the bottom left diagonal
158         boolean success = true;
159         StackNode currentNode = currentStack.getTop();
160         while (col < 9 && col > 0 && row > 0 && row < 9) {
161             for (int x = 0; x < currentStack.getCount(); x++) {
162                 if (col == currentNode.getColumn()
163                         && row == currentNode.getRow()) {
164                     System.out.println("Nope. Queen on bottom left
    diagonal");
165                     return false;
166                 }
167             }
168 
169             if (success) {
170                 col = col - 1;
171                 row = row + 1;
172             }
173         }
174 
175         return success;
176     }
177 
178     public boolean bottomRight(int col, int row) {
179         // check for queens on the bottom right diagonal
180         boolean success = true;
181         StackNode currentNode = currentStack.getTop();
182         while (col < 9 && col > 0 && row < 9 && row > 0) {
183             for (int x = 0; x < currentStack.getCount(); x++) {
184                 if (col == currentNode.getColumn()
185                         && row == currentNode.getRow()) {
186                     System.out.println("Nope. Queen on bottom right
    diagonal");
187                     return false;
188                 }
189             }
190 
191             if (success) {
192                 col = col + 1;
193                 row = row + 1;
194             }
195         }
```

```
196
197         return success;
198     }
199
200     public void showBoard() {
201         System.out.println("Drawing board...");
202         String row = "+---+---+---+---+---+---+---+---+";
203         String queen = " Q |";
204         String emptySquare = "   |";
205         String board = row;
206         String newLine = "|";
207         StackNode currentNode = currentStack.getTop();
208         boolean[][] bo = new boolean[8][8];
209         // build an array of true false values based on currentStack.
210         // true draws a queen, false draws an emptySquare
211         for (int x = 0; x < currentStack.getCount(); x++) {
212             bo[currentNode.getRow() - 1][currentNode.getColumn() - 1]
    = true;
213             currentNode = currentNode.getNext();
214         }
215
216         for (int r = 0; r < 8; r++) {
217             for (int c = 0; c < 8; c++) {
218                 if (bo[r][c] == true)
219                     newLine += queen;
220                 else
221                     newLine += emptySquare;
222             }
223
224             board = board + "\n" + newLine + "\n" + row;
225             newLine = "|";
226         }
227
228         System.out.println(board);
229     }
230
231 }
232
```