```java
package edu.ics211.h09;

import java.util.ArrayList;

/**
 * Class for recursively finding a solution to a Hexadecimal Sudoku problem.
 *
 * @author Biagioni, Edoardo, Cam Moore
 *     date August 5, 2016
 *     missing solveSudoku, to be implemented by the students in ICS 211
 */
public class HexadecimalSudoku {

  /**checks if the puzzle is full.
   * @author khj
   * @param sudoku the puzzle
   * @return true if puzzle is filled
   */
  public static boolean isFilled(int[][] sudoku) {
    for (int i = 0; i < sudoku.length; i++) {
      for (int j = 0; j < sudoku[i].length; j++) {
        int cell = sudoku[i][j];
        if (cell == -1) {
          return false;
        }
      }
    }
    return true;
  }

  /**
   * Find an assignment of values to sudoku cells that makes the sudoku valid.
   * @author khj
   * @param sudoku the sudoku to be solved.
   * @return whether a solution was found if a solution was found, the sudoku is
   *         filled in with the solution if no solution was found, restores the
   *         sudoku to its original value.
   */

  @SuppressWarnings("unused")
  public static boolean solveSudoku(int[][] sudoku) {
    //base case: sudoku is full
    if (isFilled(sudoku)) {
      return checkSudoku(sudoku, false);
    }

    // check for cell with least legal values
    int minrow = 0;
    int mincol = 0;
    int min = 16;

    for (int m = 0; m < sudoku.length; m++) {
      for (int n = 0; n < sudoku[m].length; n++) {
        if (sudoku[m][n] == -1) {
          if (legalValues(sudoku, m, n).size() < min) {
            min = legalValues(sudoku, m, n).size();
            minrow = m;
            mincol = n;
          }
```

```
60        }
61      }
62    }
63    // try each value
64    ArrayList<Integer> l = legalValues(sudoku, minrow, mincol);
65    // mini base case: if there is a cell with no legal values
66    if (l.size() == 0) {
67      return false;
68    }
69    for (int c = 0; c < l.size(); c++) {
70      sudoku[minrow][mincol] = l.get(c);
71      if (solveSudoku(sudoku)) {
72        return true;
73      }
74    }
75    sudoku[minrow][mincol] = -1;
76    return false;
77  }
78
79
80  /**
81   * Find the legal values for the given sudoku and cell.
82   *
83   * @param sudoku the sudoku being solved.
84   * @param row the row of the cell to get values for.
85   * @param column the column of the cell.
86   * @return an ArrayList of the valid values.
87   */
88  public static ArrayList<Integer> legalValues(int[][] sudoku, int row, int column) {
89    // TODO: Implement this method. You may want to look at the checkSudoku method
90    // to see how it finds conflicts.
91
92    if (sudoku[row][column] != -1) {
93      return null;
94    }
95
96    int temp = sudoku[row][column];
97
98    ArrayList<Integer> legal = new ArrayList<Integer>();
99    for (int v = 0; v <= 15; v++) {
100     sudoku[row][column] = v;
101     if (checkSudoku(sudoku, false)) {
102       legal.add(v);
103     }
104   }
105   sudoku[row][column] = temp;
106   return legal;
107 }
108
109
110 /**
111  * checks that the sudoku rules hold in this sudoku puzzle. cells that contain
112  * 0 are not checked.
113  *
114  * @param sudoku the sudoku to be checked.
115  * @param printErrors whether to print the error found, if any.
116  * @return true if this sudoku obeys all of the sudoku rules, otherwise false.
117  */
118 public static boolean checkSudoku(int[][] sudoku, boolean printErrors) {
```

```
119     if (sudoku.length != 16) {
120       if (printErrors) {
121         System.out.println("sudoku has " + sudoku.length + " rows, should have 16");
122       }
123       return false;
124     }
125     for (int i = 0; i < sudoku.length; i++) {
126       if (sudoku[i].length != 16) {
127         if (printErrors) {
128           System.out.println("sudoku row " + i + " has "
129               + sudoku[i].length + " cells, should have 16");
130         }
131         return false;
132       }
133     }
134     /* check each cell for conflicts */
135     for (int i = 0; i < sudoku.length; i++) {
136       for (int j = 0; j < sudoku.length; j++) {
137         int cell = sudoku[i][j];
138         if (cell == -1) {
139           continue; /* blanks are always OK */
140         }
141         if ((cell < 0) || (cell > 16)) {
142           if (printErrors) {
143             System.out.println("sudoku row " + i + " column " + j
144                 + " has illegal value " + String.format("%02X", cell));
145           }
146           return false;
147         }
148         /* does it match any other value in the same row? */
149         for (int m = 0; m < sudoku.length; m++) {
150           if ((j != m) && (cell == sudoku[i][m])) {
151             if (printErrors) {
152               System.out.println("sudoku row " + i + " has " + String.format("%X", cell)
153                   + " at both positions " + j + " and " + m);
154             }
155             return false;
156           }
157         }
158         /* does it match any other value it in the same column? */
159         for (int k = 0; k < sudoku.length; k++) {
160           if ((i != k) && (cell == sudoku[k][j])) {
161             if (printErrors) {
162               System.out.println("sudoku column " + j + " has " + String.format("%X", cell)
163                   + " at both positions " + i + " and " + k);
164             }
165             return false;
166           }
167         }
168         /* does it match any other value in the 4x4? */
169         for (int k = 0; k < 4; k++) {
170           for (int m = 0; m < 4; m++) {
171             int testRow = (i / 4 * 4) + k; /* test this row */
172             int testCol = (j / 4 * 4) + m; /* test this col */
173             if ((i != testRow) && (j != testCol) && (cell == sudoku[testRow][testCol])) {
174               if (printErrors) {
175                 System.out.println("sudoku character " + String.format("%X", cell) + " at row
    "
176                     + i + ", column " + j + " matches character at row " + testRow + ", column
```

```
"
                    + testCol);
              }
              return false;
            }
          }
        }
      }
    }
    return true;
  }


  /**
   * Converts the sudoku to a printable string.
   *
   * @param sudoku the sudoku to be converted.
   * @param debug whether to check for errors.
   * @return the printable version of the sudoku.
   */
  public static String toString(int[][] sudoku, boolean debug) {
    if ((!debug) || (checkSudoku(sudoku, true))) {
      String result = "";
      for (int i = 0; i < sudoku.length; i++) {
        if (i % 4 == 0) {
          result = result + "+---------+---------+---------+---------+\n";
        }
        for (int j = 0; j < sudoku.length; j++) {
          if (j % 4 == 0) {
            result = result + "| ";
          }
          if (sudoku[i][j] == -1) {
            result = result + "  ";
          } else {
            result = result + String.format("%X", sudoku[i][j]) + " ";
          }
        }
        result = result + "|\n";
      }
      result = result + "+---------+---------+---------+---------+\n";
      return result;
    }
    return "illegal sudoku";
  }
}
```