```java
package Solver;

/** This class solves a sudoku puzzle selected by the user
 *  using a 'brute force' method.
 *
 * @author Matt
 *
 * @version 1.0 (March 2013)                               */

import BasicIO.*;

public class PuzzleSolver {

    private ASCIIDisplayer d;
    private ASCIIDataFile f;

    private int[][] p = new int[9][9]; // array to map puzzle
    private int[] s; // array of guesses
    private int varSpaces = 0; // number of variable spaces (indexes in puzzle with 0)

    private boolean[] sLoc = new boolean[81]; // array to keep track of which spaces are variable

    public PuzzleSolver ( ) {

        d = new ASCIIDisplayer (15, 50);
        f = new ASCIIDataFile ();

    } // constructor

    public void solveP ( ) {

        mapP();
        if (bruteForce()) {
            d.writeLine ("Puzzle successfully solved!");
            d.writeLine ("");
            d.writeLine ("Solved puzzle:");
            for (int i = 0 ; i < 9 ; i++) {
                for (int j = 0 ; j < 9 ; j++) {
                    d.writeInt(p[j][i]);
                } // for
                d.writeLine("");
            } // for
        } else {
            d.writeLine ("Puzzle could not be solved.");
        } // else


    } // solveP

    private void mapP ( ) { // maps sudoku puzzle to array

        int k = 0;

        d.writeLine ("Mapping sudoku puzzle to array...");

        for (int i = 0 ; i < 81 ; i++) { // initializes sLoc variable (false = static, true =
variable)
```

```
            sLoc[i] = false;
        } // for

        for (int i = 0 ; i < 9 ; i++) {
            for (int j = 0 ; j < 9 ; j++) {
                p[j][i] = f.readInt(); // read in board to 2-dimensional array
                if (p[j][i] == 0) { // if location is blank (0), set as variable (squares
numbered from 1 to
                    sLoc[k] = true; // 81, from the top left to bottom right
                    varSpaces++;
                } // if
                k++;
            } // for
        } // for
        f.close();

        s = new int[varSpaces]; // create array size of variable spaces

        for (int i = 0 ; i < varSpaces ; i++) { // initialize array with lowest possible integers
            s[i] = 1;
        } // for

    } // mapP

    private boolean bruteForce ( ) {

        d.writeLine ("Attempting to solve via brute force...");
        if (generate(varSpaces-1)) {
            d.writeLine ("Done!");
            return true;
        } else {
            d.writeLine ("An error occured...");
            return false;
        } // else

    } // bruteForce

    private boolean generate ( int m ) { // stackOverflow on easy/hard puzzle, works fine on
easyeasy puzzle

        int k = 0;
        int c = 0;

        for (int i = 0 ; i < 9 ; i++) {
            for (int j = 0 ; j < 9 ; j++) {
                if (sLoc[k] == true) { // if location is variable, replace with associated guess
                    p[j][i] = s[c];
                    c++;
                } // if
                k++;
            } // for
        } // for

        if (valid()) {
            return true;
        } else {
            if (s[m] < 9) {
                s[m]++;
                if (m != varSpaces-1) {
                    m++;
                } // if
                generate(m);
            } else {
                s[m] = 1;
                if (m != 0) {
                    m--;
                    generate(m);
                }
                return false;
            } // else
        } // else

        return false;

    } // generate

    private boolean valid () {

        boolean[] numCheck = new boolean[9];
        boolean isValid = true;

        for (int i = 0 ; i < 9 ; i++) { // initialize numCheck (false = unused, true = used)
            numCheck[i] = false;
        } // for

        for (int i = 0 ; i < 9 ; i++) { // check if solved vertically
            for (int j = 0 ; j < 9 ; j++) {
                if (numCheck[p[i][j]-1] == true) {
                    isValid = false;
                } else {
                    numCheck[p[i][j]-1] = true;
                } // else
            } // for
            for (int j = 0 ; j < 9 ; j++) { // reinitializes numCheck
                numCheck[j] = false;
            } // for
        } // for

        if (isValid == true) { // only checks if valid
            for (int i = 0 ; i < 9 ; i++) { // check if solved horizontally
                for (int j = 0 ; j< 9 ; j++) {
                    if (numCheck[p[j][i]-1] == true) {
                        isValid = false;
                    } else {
                        numCheck[p[j][i]-1] = true;
                    } // else
                } // for
                for (int j = 0 ; j < 9 ; j++) { // reinitializes numCheck
                    numCheck[j] = false;
                } // for
            } // for
        } // if

        if (isValid == true) { // only checks if valid
```

```
            for (int i = 0 ; i < 3 ; i++) {
                for (int j = 0 ; j < 3 ; j++) {
                    for (int k = 0 ; k < 3 ; k++) {
                        for (int l = 0 ; l < 3 ; l++) {
                            if (numCheck[p[k+(3*i)][l+(3*j)]-1] == true) {
                                isValid = false;
                            } else {
                                numCheck[p[k+(3*i)][l+(3*j)]-1] = true;
                            } // else
                        } // for
                    } // for
                    for (int k = 0 ; k < 9 ; k++) { // reinitializes numCheck
                        numCheck[k] = false;
                    } // for
                } // for
            } // for
        } // if
        return isValid;

    } // valid

    public static void main ( String[] args ) {new PuzzleSolver().solveP(); };

} // PuzzleSolver
```