```java
import BasicIO.*;

/**
 * This program will recursively solve any maze in the format of mz1.txt or
 * provided that it can be solved.
 *
 * @author Matt Laidman
 * @version 1.0 (March 2014)
 */
public class MazeWalk {

    private char[][] board; // maze
    private boolean[][] checked; // is space in maze checked
    private int sX, sY, mRows, mCols; // start x, start y, max rows, max cols
    private ASCIIDataFile maze; // maze file
    private boolean solved = false; // is maze solved

    public MazeWalk ( ) {
        getStart(); // get start location/build entry form
        getMaze();
        solve();
    }


    private void findPath (int row, int col) {
        if (row < 0 | row >= mRows | col < 0 | col >= mCols) { // if move is out of bounds
            return;
        }
        if (board[row][col] == 'V' | board[row][col] == '>' | board[row][col] == '<' | board[row][col] == '^') { // if move is already made
            return;
        }
        if (board[row][col] == '#') { // if move is maze wall
            return;
        }
        if (checked[row][col]) { // if move has already been checked
            return;
        }
        if (board[row][col] == 'E') { // if move is exit
            solved = true; // flag maze as solved
            for (int i = 0 ; i < mRows ; i++) { // print completed maze
                for (int j = 0 ; j < mCols ; j++) {
                    System.out.print(board[i][j]);
                }
                System.out.println();
            }
        }
        checked[row][col] = true; // set space as checked
        board[row][col] = 'v'; // check down
        findPath(row+1, col);
        board[row][col] = '>'; // check right
        findPath(row, col+1);
        board[row][col] = '<'; // check left
        findPath(row, col-1);
        board[row][col] = '^'; // check up
        findPath(row-1, col);
        board[row][col] = ' '; // return to blank if not path
    }


    private void solve ( ) {
        checked = new boolean[mRows][mCols]; // set all spaces to unchecked
        for (int i = 0 ; i < mRows ; i++) {
            for (int j = 0 ; j < mCols ; j++) {
                checked[i][j] = false;
            }
        }
        findPath(sX, sY); // attempt to find path
        if (solved) {
            System.out.println("Maze successfully solved!");
        } else {
            System.out.println("Maze could not be solved.");
        }
    }


    private void getMaze ( ) {
        mRows = Integer.valueOf(maze.readString()); // get max rows
        mCols = Integer.valueOf(maze.readString()); // get max columns
        board = new char[mRows][mCols]; // read in board
        for (int i = 0 ; i < mRows ; i++) {
            for (int j = 0 ; j < mCols ; j++) {
                board[i][j] = maze.readC();
                if (board[i][j] == 10) board[i][j] = maze.readC(); // if new line character, get next char
            }
        }
    }


    private void getStart ( ) {
        BasicForm start = new BasicForm(); // build form
        start.addTextField("sX", "X");
        start.addTextField("sY", "Y");
        start.addRadioButtons("mSelect", "Select Maze: ", false, "Small", "Large");
        start.addLabel("label", "Leave blank to select your own maze.");
        start.accept();
        sX = start.readInt("sX"); // get start location
        sY = start.readInt("sY");
        if (start.readInt("mSelect") == 1) { // get selected maze
            maze = new ASCIIDataFile("mz2.txt");
        } else if (start.readInt("mSelect") == 0) {
            maze = new ASCIIDataFile("mz1.txt");
        } else {
            maze = new ASCIIDataFile();
        }
        start.close();
    }


    public static void main(String[] args) {
        new MazeWalk();
    }
}
```