```
package hw3;

import static api.Orientation.*;
import static api.CellType.*;

import java.util.ArrayList;

import api.Cell;

/**
 * Utilities for parsing string descriptions of a grid.
 * @author jcluse
 */
public class GridUtil {
      /**
       * Constructs a 2D grid of Cell objects given a 2D array of cell
descriptions.
       * String descriptions are a single character and have the following meaning.
       * <ul>
       * <li>"*" represents a wall.</li>
       * <li>"e" represents an exit.</li>
       * <li>"." represents a floor.</li>
       * <li>"[", "]", "^", "v", or "#" represent a part of a block. A block is not
a
       * type of cell, it is something placed on a cell floor. For these
descriptions
       * a cell is created with CellType of FLOOR. This method does not create any
       * blocks or set blocks on cells.</li>
       * </ul>
       * The method only creates cells and not blocks. See the other utility method
       * findBlocks which is used to create the blocks.
       *
       * @param desc a 2D array of strings describing the grid
       * @return a 2D array of cells the represent the grid without any blocks
present
       */
      public static Cell[][] createGrid(String[][] desc) {

            Cell[][] grid = new Cell[desc.length][desc[0].length];

                  for(int i = 0; i < desc.length; i++) {

                        for(int j = 0; j < desc[i].length; j++) {

                              String str = desc[i][j];

                              if (str.equals("*")) {

                                    grid[i][j] = new Cell(WALL, i, j);

                              }
                              else if (str.equals("e")) {

                                    grid[i][j] = new Cell(EXIT, i, j);

                              }
                              else {

                                    grid[i][j] = new Cell(FLOOR, i, j);
```

```
                        }
                    }
                }
                return grid;
        }

        /**
         * Returns a list of blocks that are constructed from a given 2D array of
cell
         * descriptions. String descriptions are a single character and have the
         * following meanings.
         * <ul>
         * <li>"[" the start (left most column) of a horizontal block</li>
         * <li>"]" the end (right most column) of a horizontal block</li>
         * <li>"^" the start (top most row) of a vertical block</li>
         * <li>"v" the end (bottom most column) of a vertical block</li>
         * <li>"#" inner segments of a block, these are always placed between the
start
         * and end of the block</li>
         * <li>"*", ".", and "e" symbols that describe cell types, meaning there is
not
         * block currently over the cell</li>
         * </ul>
         *
         * @param desc a 2D array of strings describing the grid
         * @return a list of blocks found in the given grid description
         */
        public static ArrayList<Block> findBlocks(String[][] desc) {

              ArrayList<Block> blocks = new ArrayList<Block>();

              for(int row = 0; row < desc.length; row++) {

                    for(int col = 0; col < desc[row].length; col++) {

                          if (desc[row][col].equals("[")) {
                                int endCol = col + 1;
                                while (!desc[row][endCol].equals("]")) {
                                      endCol++;
                                }
                          Block block = new Block(row, col, endCol - col + 1,
HORIZONTAL);

                          blocks.add(block);

                          }

                          else if (desc[row][col].equals("^")) {
                                int endRow = row + 1;
                                while (!desc[endRow][col].equals("v")) {
                                      endRow++;
                                }
                          Block block = new Block(row, col, endRow - row + 1,
VERTICAL);

                          blocks.add(block);

                          }
```

```
                }

            }
            return blocks;
        }

    }
```