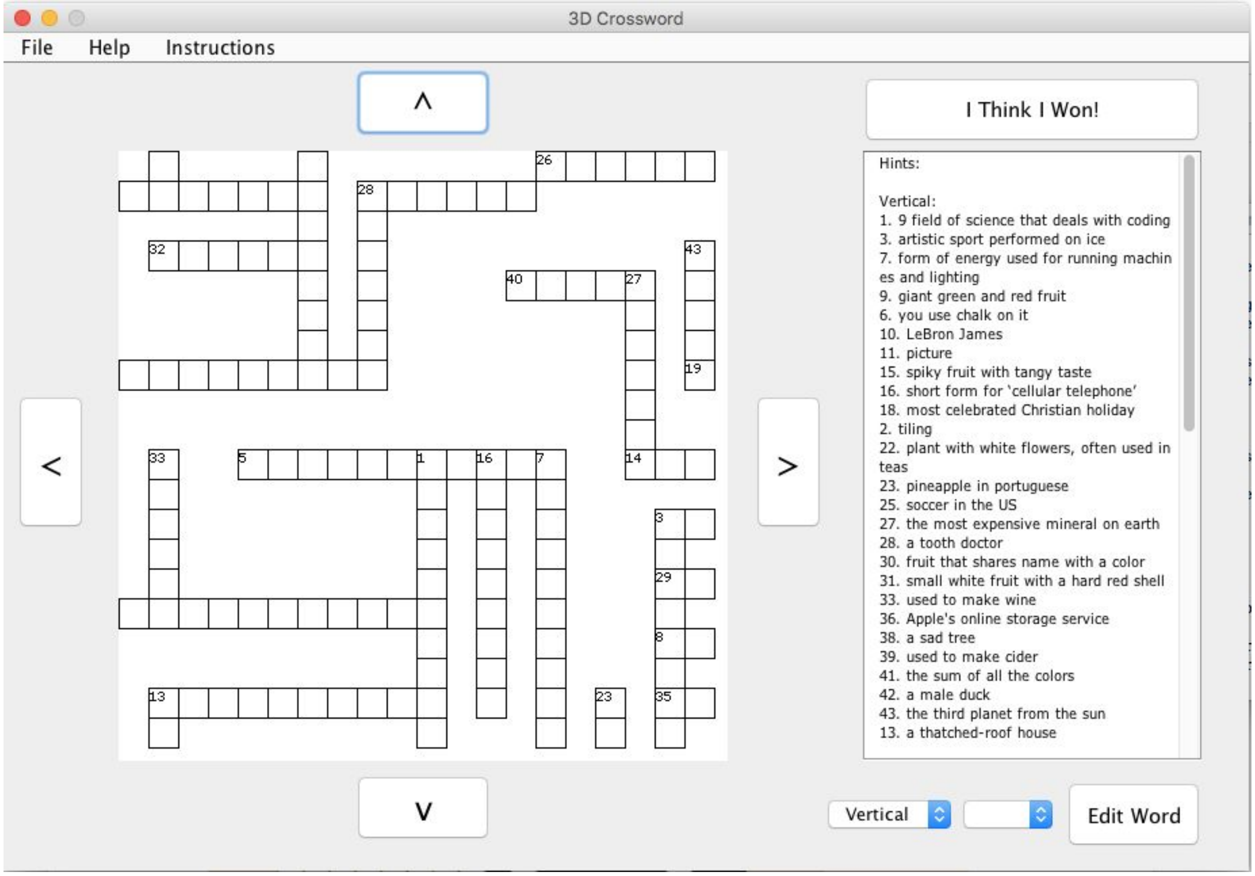
Pseudo-3D Crossword

Pedro B. Balduino Morais, Arnav Parikh, Drake Kufwafwa, Harris Khawar

Part 1 - High Level Design

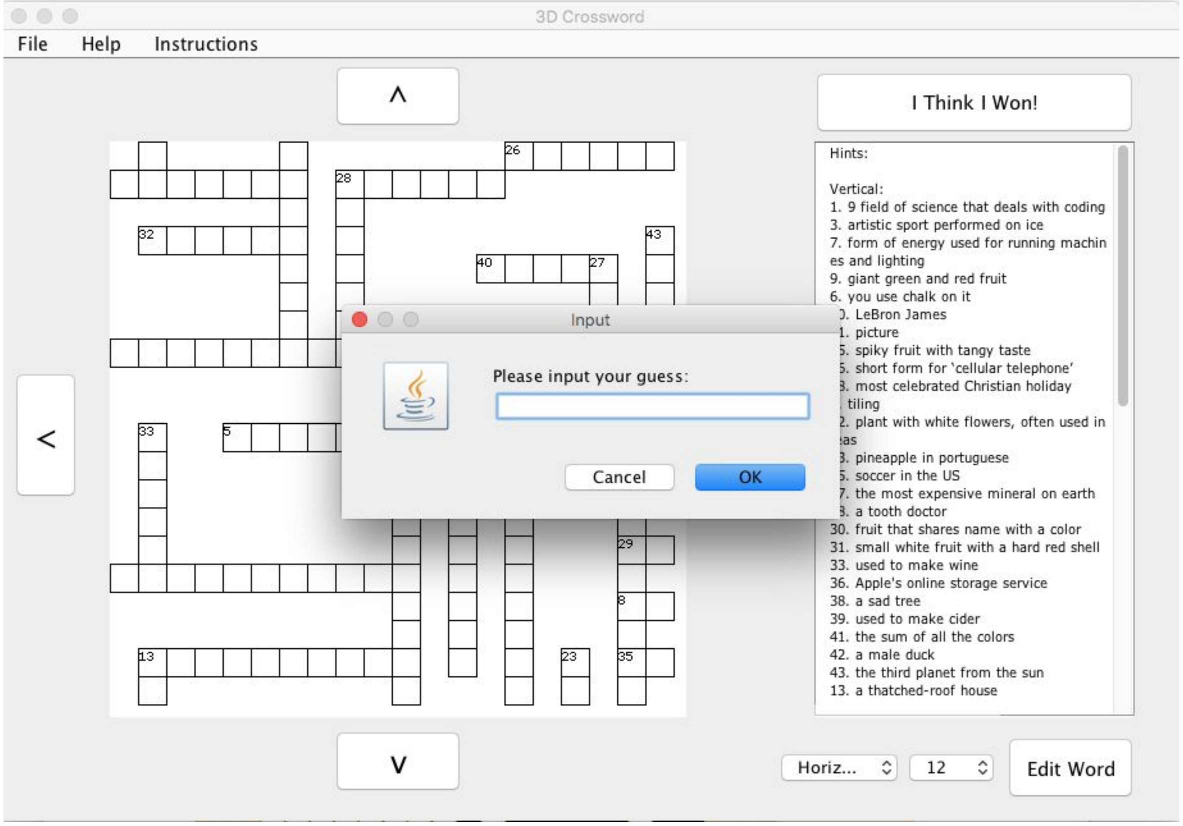
What our program looks like:

Our program operates on its own window. It consists of a menu bar, a graphical display for the crossword, a text display for the hints, and several buttons that are used to navigate the different aspects of solving this pseudo-3D crossword. The buttons consist of four arrow buttons surrounding the grid, used for switching perspectives; a button on the bottom right and two drop down menus, used to select a word to edit; and a button on the top right, used to check if the puzzle is complete. Pressing the different buttons leads to different pop-up menus that are used to display messages to the user and receive user input. Some screenshots are included below:



Hints:

Vertical:
1. 9 field of science that deals with coding
3. artistic sport performed on ice
7. form of energy used for running machines and lighting
9. giant green and red fruit
6. you use chalk on it
10. LeBron James
11. picture
15. spiky fruit with tangy taste
16. short form for 'cellular telephone'
18. most celebrated Christian holiday
2. tiling
22. plant with white flowers, often used in teas
23. pineapple in portuguese
25. soccer in the US
27. the most expensive mineral on earth
28. a tooth doctor
30. fruit that shares name with a color
31. small white fruit with a hard red shell
33. used to make wine
36. Apple's online storage service
38. a sad tree
39. used to make cider
41. the sum of all the colors
42. a male duck
43. the third planet from the sun
13. a thatched-roof house

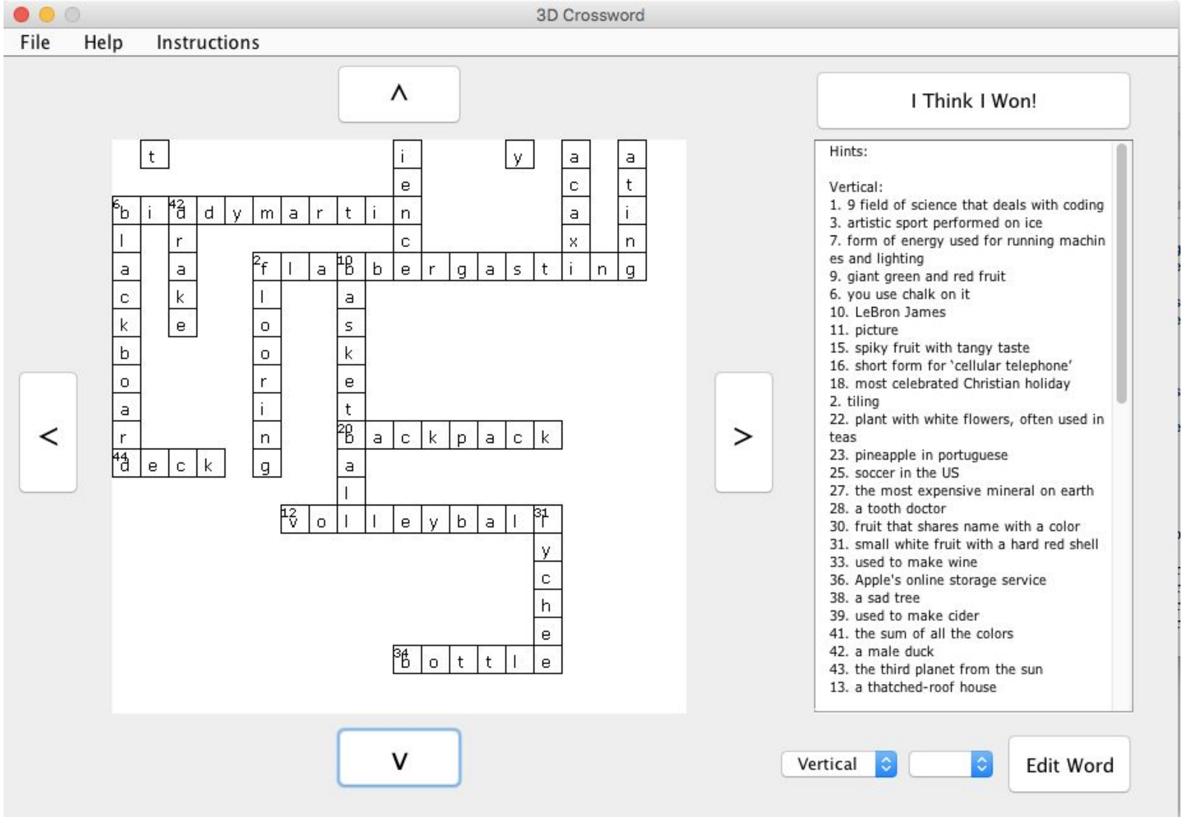


Please input your guess:

Hints:

Vertical:
1. 9 field of science that deals with coding
3. artistic sport performed on ice
7. form of energy used for running machines and lighting
9. giant green and red fruit
6. you use chalk on it
). LeBron James
1. picture
5. spiky fruit with tangy taste
5. short form for 'cellular telephone'
3. most celebrated Christian holiday
tiling
2. plant with white flowers, often used in
as
3. pineapple in portuguese
5. soccer in the US
7. the most expensive mineral on earth
3. a tooth doctor
30. fruit that shares name with a color
31. small white fruit with a hard red shell
33. used to make wine
36. Apple's online storage service
38. a sad tree
39. used to make cider
41. the sum of all the colors
42. a male duck
43. the third planet from the sun
13. a thatched-roof house

Horiz... 12



Hints:

Vertical:
1. 9 field of science that deals with coding
3. artistic sport performed on ice
7. form of energy used for running machines and lighting
9. giant green and red fruit
6. you use chalk on it
10. LeBron James
11. picture
15. spiky fruit with tangy taste
16. short form for 'cellular telephone'
18. most celebrated Christian holiday
2. tiling
22. plant with white flowers, often used in teas
23. pineapple in portuguese
25. soccer in the US
27. the most expensive mineral on earth
28. a tooth doctor
30. fruit that shares name with a color
31. small white fruit with a hard red shell
33. used to make wine
36. Apple's online storage service
38. a sad tree
39. used to make cider
41. the sum of all the colors
42. a male duck
43. the third planet from the sun
13. a thatched-roof house

Vertical

How user input works:

Firstly, the user must specify a .txt file as the System input, when running the program from the command line.

Once the program generates the crossword, the main user input consists on clicking on the various aspects of the user interface: menu bars, buttons and drop down menus.

Another form of user input is text entered from the keyboard. This is available after the user presses the "Edit Word" button, once a word has been selected from the drop down menus. The user then is able to type in their guesses for the words in the puzzle.

How the programs responds to user input:

User specifies file on input:
The program will respond to receiving a specific file by parsing it into the separate words and their respective definitions, and then storing these into different instances of the Word class. The program will then generate an array containing the list of instances of Word. This list is then used to generate the crossword puzzle.

User presses one of the switch perspective buttons:
The program will switch the current display boundaries to the display boundaries of the grid section adjacent in the direction of the button, if such a section exists.

User presses the Edit Word button:
If a word is selected from the drop down menus to the left, the program will display a pop-up that prompts the user to enter his guess for the word. If such a guess is valid, the program will then display the guess on the grid. If such a guess is invalid, the program will display a message telling the user why.

User presses the I Think I Won! button:
The program will run a test to check if the puzzle is correctly completed or not, and will inform the user with a pop-up message whatever is the case.

User presses a menu bar item:
Pressing on the various menu bar items will cause each item to carry out its unique functionality. For instance, "Show Instructions" will display usage instructions, while "About 3D Crossword" will display information about the program.

The purpose our program serves:

The program is, in essence, a game - it creates a puzzle, which the user must then attempt to solve. It is intended to be used for the sheer excitement of generating and solving pseudo-3D crossword puzzles. These can be generated by yourself or by your friends, or even from word lists found online.

Part 2 - Detailed Design Description

Class UserInterface:

This is "main" class that actually runs the program. As such, there is no specific instance of this class created. This class holds an instance of Crossword, which represents the puzzle to be solved. It is also responsible for handling the user interface and user input, including graphics, as well as the general progression of the program. Finally, it is also responsible for handling the file input and parsing it into the words and hints necessary for the puzzle.

Fields:
All fields in this class are private because this class is self-contained, and is not supposed be used in any other classes.

- Several JComponent declarations, used to define the structure of the programs user interface.
  - These represent the different elements of the user interface.
- Word[] wordList
  - This contains and represents the list of words that is used to generate the crossword.
- Crossword puzzle
  - This instance of the Crossword class represents the puzzle generated from the wordList. It represents the puzzle grid and the words that are ultimately stored in it.
- Character[ ][ ] userArray
  - This field represents the grid of characters inputted by the user, as opposed to the actual solution to the puzzle. This is mainly used for graphical representation of the user input.
- LinkedBounds currentBounds
  - This field represents the boundaries of the current section of the puzzle grid being displayed. It is used to switch between perspectives and graphically change the grid.
- LinkedBounds initialBounds

    - The same as currentBounds, but stores the bounds that were initially displayed.

Constructor:

- There is only one constructor, as the only reason it exists is to initialize and display all user interface elements using java.awt and java.swing functionalities.

Methods:

All methods except main are private because the class is self contained and is not intended to be used in any other classes.

- public static void main(String [ ] args)
    - Standard initialization of main method. The main method in this class is responsible for reading the list of word/definitions from the input file and storing it as a list of Word objects. Furthermore, the method handles initialization for the aspects of the program such as the puzzle, the hints and the graphical display.
- void sendWordSelection( )
    - Handles the creation of the drop down menus for word selection, relating to the orientation and hint number of the word. Links the two so that hint number selection depends on orientation. The values in these two menus are then used to search for the desired word.
- void sendMenuBar( )
    - Handles the creation of the menu bar and its options. Most important functionalities added are detailing the instructions of the program and one for solving the puzzle automatically. This allows quick testing and avoids frustration.
- void sendButtons( )
    - Handles the creation of the button and their functionalities. For each button:
        - Perspective buttons: Makes so each button switches the current perspective displayed of the grid. Does this by changing which current boundaries of the grid are currently displayed, using the currentBounds field.
        - Edit word button: Allows the user to edit a Words userArray, and edits the corresponding area of the userGrid. Does this by taking in user text input and converting it into the necessary character arrays. Only works if a word is currently selected in the drop down menus.
        - I Think I Won Button: Clicking this button causes the program to check if the user has won by calling the checkVictory() method in the Crossword class. It then displays a message confirming or denying that the user has won.
- void sendDisplay( )
    - Handles the creation of the graphical display area and the hint display area, setting where in the user interface they are located. Gets list of hints from the puzzle field.
- void sendUI( )

    - This method handles the syntax necessities of java.awt, and initializes all the necessities for the user interface.
- static void initializeBounds( )
    - Creates the five instances of the LinkedBounds class, that are used to represent the different sections of the grid to be displayed. This method also sets up the links between them.
- static boolean startisInBounds(Word toCheck)
    - This method takes in a Word object, and returns a boolean that specifies whether or not that word's first character is located within the bounds of the current display. Used for displaying the hint numbers in the grid.
- class graphicsFrame extends JPanel
    - Is an interior class that provides a customized JPanel to be used in the UI. The two fields are its width and height, and the sole constructor sets these to be 450. The important this in this class is the method paintComponent:
    - public void paintComponent(Graphics g)
        - This method follows the usual overriding for paintComponent, and is responsible for painting the gridSquares, the hint Numbers and the words inputted by the user in the UserGrid.

Class Crossword:

This class represents an individual crossword puzzle, which includes its grid and hints. The class also keeps track of the Word objects used to create it, and user input into these objects. When the program is run, a single instance of this class is created and used in the UserInterface class.

Fields:

- static Character[ ][ ] grid
    - This Character[ ][ ] represents the grid of the crossword puzzle, with all words correctly placed in it. It is static because only one instance of Crossword is used at a time, so having it static minimized writing.
- static Word[ ] storedWords
    - This array stores the Word objects that have been placed into the grid. Note that this is not the same as the initial list of words: if certain words do not make the grid, they are not stored in this array. It is static because only one instance of Crossword is used at a time, so having it static minimized writing.
- static int placedCounter
    - This int represents the number of words that have been placed in the grid, and is mainly used for keep track of the storedWords array. It is static because only one instance of Crossword is used at a time, so having it static minimized writing.
- static int hintCounter

    - This int represents and keeps track of the current hint number to be added to a word, if a word does not share an initial position with another word. It is static because only one instance of Crossword is used at a time, so having it static minimized writing.
- static int bottomCounter, topCounter, rightCounter, leftCounter
    - These ints keep track of how many characters have been placed in each grid region, barring the middle region. This has an important use in the getScore method. They are static because only one instance of Crossword is used at a time, so having them static minimized writing.

Constructor: public Crossword (Word[ ] wordList) {...}

- Takes in an array of Words, sorts in from longest to shortest and then calls the initialize grid method using the list of Words.
- There is only one constructor because, in our usage of the class, we would only ever want the constructor to take in a wordList. Other constructors were unnecessary.
- The constructor is public because it must be called from within other classes, namely, the UserInterface class.

Methods:

All static methods are static because they only needed to be called within the Crossword class, thus minimizing writing.

- Public static void initializeGrid(Word[ ] list)
    - Does the actual initialization of the grid. First this divides the grid into 9 sections and sets the four corners to be null, essentially creating a plus-shaped grid which is “wrapped” around the cube. Then, this looks through the list of words in the argument and attempts to place each word on the list. Once a word is placed, the method loops back to the beginning of the list. This is repeated until no more words can be placed on the grid.
- Public static boolean attemptToPlace(Word toPlace)
    - This method takes in a Word and attempts to Place it in the grid. This method looks through all possible coordinates the word could be placed, assigns a score for each coordinate, and picks the one with the highest score. If there is a suitable placement location, this method will call the placeWord method on that word and location, and return true, indicating the placement was successful. Otherwise, the method will return false, indicating the placement was unsuccessful.
- private static void PlaceWord(Pair coordinates, Word word, boolean vertical)
    - This method takes in a Pair of coordinates, a word, and a boolean indicating whether the word is vertical or not. It then proceeds to place that word on the given coordinates of the grid. It assumes that placement is valid because this is accounted for in the attempt To Place method. It also handles storing the hint number for the word that is placed.
- Public static int getScore(Pair coordinates, Word word, boolean vertical)

    - This is where the magic of the algorithm happens. Given the same three arguments as the method above, this method returns an int representing a score for the position if the word were to be placed there. The method also invalidates placement if the word infringes any rules. The three conditions accounted for in the score are number of adjacent characters, number of words crossed, and number of characters placed in a surrounding region. The algorithm will prioritize words in regions with relatively few characters placed, words with relatively few adjacent characters, and words that cross more words. This ensures that the crossword spreads mostly evenly in all four directions, given an initial placement in the middle.
- Public static int getNumAdjacent(Pair position)
    - This method takes in a position in the grid and returns an int representing the number of characters placed in tiles adjacent to that position. It is used in the getScore() method.
- Public static void incrementCharCounter(Pair position)
    - This method takes in a position in the grid and increments the placed character counter for the region that contains that position. It is called in the place word method whenever a character is placed, and is essential for the getScore method to work.
- Public static int getCharCounter(Pair position)
    - This method takes in a position in the grid and returns the placed character counter for the region containing that position. It is called in the getScore method, to ensure the algorithm prioritizes relatively unpopulated areas.
- Public String getHints( )
    - This method returns a String representation of all the hints used by stored words, separated by vertical and horizontal words.
    - This method is public and non-static because it is meant to be called on the specific instance of Crossword in the UserInterface class, to get the hints and display them.
- Public Boolean checkVictory( ) {
    - This method checks if all the userArrays in each instance of Word are equivalent to the actual character arrays, meaning that the user inputs are equal to the solution. If that is true, the method should return true; otherwise, it should return false.
    - This is public and non-static because it is meant to be called on the specific instance of Crossword in the UserInterface class.
- Public static Word[ ] removeAtIndex(int index, Word[ ] myArray)
    - This is a helper method that returns a copy of the given array with the object at the specified index removed. Used to removed words that are already placed from the word list.
- Public static void printGrid( )
    - Helper method that prints out the current grid to console. Used only for testing purposes.

Class Word:

This class will represent the actual words used to generate the crossword puzzle; as such, it will contains words as well as the hints associated with them. Furthermore, this class will contain the user input associated with what the user believes the word to be. At runtime, there will be several instances of this class used in both the UserInterface class and the Crossword class. This is basically the backbone of the program.

This class implements the Comparable<Word> interface, so that instances of this class are comparable between themselves. This facilitates sorting words by enabling the sort method in java.util.Arrays.

Fields:

- Private char[ ] wordChars
    - Char array representation of the word represented by this object.
    - This is private because under no circumstances do we want to change this outside of the Word class. The actual solution to the puzzle should never change.
- Public char[ ] userArray
    - Char array representation of the word inputted by the user, what the user believes this word to be.
- Private boolean vertical
    - Boolean that represents whether a word is placed vertically within the grid or not.
    - Once again private because we do not want to alter this too much outside of the Word class.
- String hint
    - The string representation of the hint associated with this word.
- Int hintNum
    - Int that represents the hint number associated with this word.
- Pair initialCoordinates
    - Object that stores the coordinates to the word first letter on the grid. User for easy access to a word's grid position.

Constructor: public Word(String toStore)

- Takes in a String, sets it to lowerCase, converts it to a char array and stores it in this instance of word as this words wordChars.
- Only one constructor because, in our usage of the class, there are no other parameters we would want the constructor to take in, rendering other constructors useless.

Methods:

- Public boolean checkMatch( )

    - Checks if the contents of the user input and the puzzle solution for this specific word match. If so, return true; otherwise, return false.
    - This is public and non-static because it is intended to be called on each instance of Word when checking victory in the Crossword class.
- Public char[ ] getCharArray( )
    - Getter method for the char representation of the actual word. Returns a copy of the wordChars for this word for use in other classes.
    - This is public and non-static because it is intended to be called on instances of Word in other classes.
- Public boolean getVertical( )
    - Getter method for the verticality of the word. Returns the vertical boolean.
    - This is public and non-static because it is intended to be called on instances of Word in other classes.
- Public void setVertical( )
    - Sets this instance of Word's verticality to true.
    - This is public and non-static because it is intended to be called on instances of Word in other classes.
- Public void setHorizontal( )
    - Sets this instance of Word's verticality to false.
    - This is public and non-static because it is intended to be called on instances of Word in other classes.
- Public boolean sharesChar(Word toCompare)
    - Takes in a word and check if this instance shares any characters with the word argument. If so, return true; otherwise, return false. Intended for finding possible positions where words can be placed.
    - This is public and non-static because it is intended to be called on instances of Word in the attemptToPlace method in the Crossword Class.
- Public int compareTo(Word o)
    - Overriding the interface method, implemented so instances of Word can be compared. Compares words based on their length. If this instance of word is shorter, return a positive int. If it is equal, return 0. If it is longer, return a negative int.
- Public String toString( )
    - Helper method for testing purposes. Returns a string representation of this word, containing both the word itself and its hint.

Class LinkedBounds:

This Class is a data structure that holds a group of boundaries that represents a certain section of the puzzle grid. What classifies it as a data structure is that it holds the links between the adjacent sections of a grid, given they are mapped onto faces of a cube. When the program is running, there are five instances of this Class created, each representing a “face” of the top and sides of a cube. This is the data structure that allows the switching of perspective between sections of the grid.

Fields:

- Public int minX
    - This int represents the minimum x coordinate of the boundary
- Public int minY
    - This in represents the minimum y coordinate of the boundary
- Public LinkedBounds leftLink, rightLink, topLink, bottomLink
    - These LinkedBounds represent the other sections of the grid that are linked to this one; think of each as the faces of the cube that are adjacent to the current face.

Constructor:

- Public LinkedBouns(int x, int y)
    - Sets the passed in values to be the boundaries minX and minY, respectively.
    - Only one constructor because accounting for more arguments was not necessary.

Methods:

- Public void setLinks(LinkedBounds left, LinkedBounds right, LinkedBounds top, LinkedBounds bottom)
    - Sets the arguments to be the left, right, top and bottom links for this instance of LinkedBounds, respectively.
    - This is public and non-static because it is intended to be called on instances of LinkedBounds in the UserInterface class.

Class Pair

This is a helper class that represents a pair of coordinates, and made the passing into methods and storing into classes of coordinates very convenient.

Fields:

- Int x
    - Int that represents the x coordinate for this pair.
- Int y
    - Int that represents the y coordinate for this pair.

Constructor: public Pair(int x, int y)

- Stores the arguments passed in into the x and y fields, respectively.
- Only one constructor was necessary, as only this pair of arguments was necessary.

Structure of the program:

The User Interface class contains all the user interface initialization and generation. In order to correctly display the section of the grid, it utilizes the LinkedBounds data structure, containing five instances of LinkedBounds that are linked between each other. Furthermore, it generates and holds an instance of the Crossword class, which generates and contain the puzzle grid. Finally, it generates an Array of Word objects from the user input, which it then passes on to the Crossword class it contains.

The Crossword class handles all the nitty-gritty aspects of handling the several Word objects and plotting them onto a grid. It achieves this by utilizing a placement algorithm that scores each possible position a word can be placed, and placing it in the position with the highest score. It also utilizes the compactness of the Pair class to aid in working with the grid's coordinate system.

Basically, we can think of the Word, Crossword and UserInterface classes as zooming in and out of the scope of the program. Zoom in and you see how it is all structured by words, which are the building blocks that build the Crossword. Zoom out again and you'll see how the Crossword is used to build the UserInterface. This design was thus very intuitive and easy to work with on a higher level.