

## Cross Word Grid

---

### Due Date:

This assignment is due Friday 2023-03-24.

---

### Purpose:

To gain practice with the `javax.swing.*` architecture, in particular, using fonts and manual object layouts.

This assignment also has an algorithmic component: determining where crossword words start and end. Providing appropriate word numbers requires some thought.

---

### The Problem

Write a “cross-word” generating program that takes input similar to that shown in Figure 1, and produces output similar to that shown in Figure 2.

The program should read from an input file and produce a GUI.

The format of the input file is illustrated in Figure 1. The first two lines of an input file are numbers: the first number is the number of rows ( $r$ ) in the crossword and the second number is the number of columns ( $c$ ). After that come  $r$  lines of text, each line containing exactly  $c$ -characters. The last  $r$  lines of text consist only of alphabetic characters, all of which are either lower-case letters or capital 'X'.

The format of the output GUI is illustrated in Figure 2. It consists, at least conceptually, of three components.

The first component is an  $r \times c$  grid of squares that are either black or white. Xs on the input are represented by black squares on the output. Other characters are drawn in white squares in uppercase. White squares have a small number in the top-left corner if they are the beginning of either a horizontal or vertical word. (It is possible that they may be both. See the square numbered 4 in the example.) One-letter words are ignored (in the example, the A in the first column is not considered a vertical word).

The large letters should be centered if possible. (See below.)

The second component is a column of the horizontal words with their starting numbers. The third component is a column of the vertical words with their starting numbers.

---

```
in.txt
5
4
Xera
chat
aXXX
theX
sell
```

---

Figure 1: Sample Input

---

## A Possible Method

The following advice is recommended advice only. If you find other ways to implement the requested functionality, please feel free to do so.

### The Word Lists

The second and third components of the display are most easily implementable as JTextAreas. To use a different font for the titles, make the title a JLabel, and combine it with the word list using a JPanel with a BorderLayout layout.

### The Grid

The grid in the first component is the hardest component to display. One approach that is possible here is to subclass JPanel, and then use `setLayout(null)` to remove the layout manager, and then explicitly set the location of each object that you wish to display, à la

```
add(mySubObject) ;
mySubObject.setLocation(x, y) ;
```

To get the grid itself to take up enough space in the overall frame, you may need to override the `getMinimumSize` method.

The individual cells of the can be subclassed from JComponent. If you use `setBorder`, it is easy to get the black line around them. In this case you want to override the

```
protected void paintComponent(Graphics g)
```

method in order to the paint the cell. Remember to call `super.paintComponent(g)` first, and then do whatever painting you need. You probably want to do a `g.setColor` one or more times, a `g.setFont` zero or more times, and then various `g.fillRect`'s and `g.drawString`s. The graphics object that you get *has co-ordinates relative to the cell, not the whole frame*.

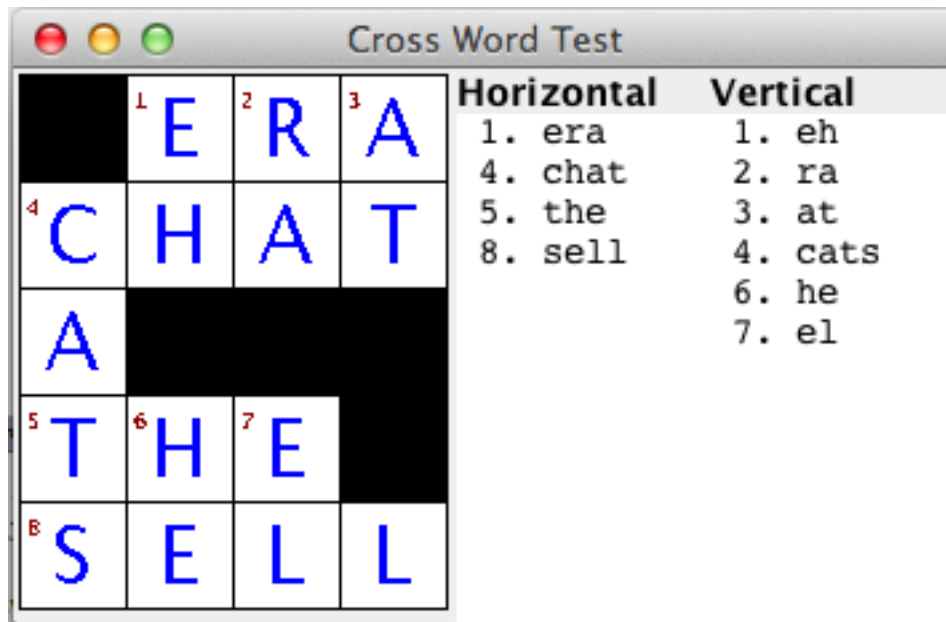


Figure 2: Sample Output

## Centering Letters

To center characters in a cell, you will want to use a `FontMetrics` object.

```
FontMetrics fm = graphics.getFontMetrics(getFont()) ;
... fm.charWidth(myChar);
... fm.getStringBounds(""+myChar,0,1,g2d).getHeight() ;
... fm.getDescent()
```

The `FontMetrics` object is designed for getting measurements of a character shape. The *descent* of a font measures how much characters can fall below the baseline. The third argument of the `graphics.drawString(s,x,y)` function specifies the baseline for the string, not the bottom of its bounding rectangle.

(See Figure 3 on the following page. *width* there corresponds to `.charWidth(myChar)`; *yMax* corresponds to `.getHeight()`; and *-yMin* to `.getDescent()`.)

In Figure 2, the crossword grid square numbers are set in a sans serif font at 8 points with colour (140,0,0), and the letters are set in a sans serif font at 30 points with colour blue. The frame reflects the fact that the program was running under Mac OS 10.9.5. Other OS's will look slightly different.

## Submission

As usual.

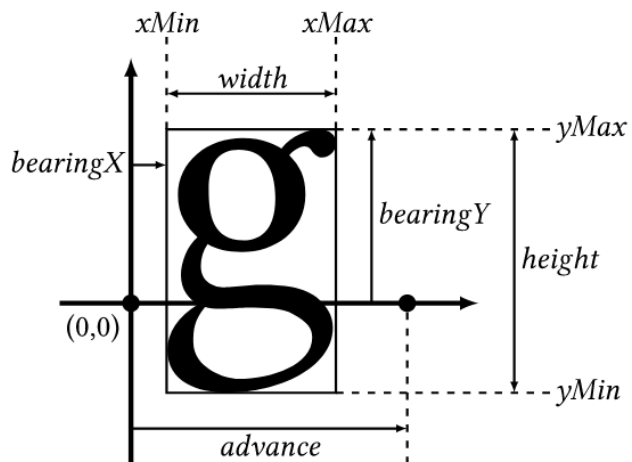


Figure 3: Font metric illustration from <https://freetype.org/freetype2/docs/glyphs/glyphs-3.html>