# Cross Word Grid

## Due Date:

This assignment is due Friday, March 15.

## Purpose:

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

## Cross Word Grid

Write a "cross-word" generating program that takes input similar to that shown in Figure 1 on page 3, 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. `X`s 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 1 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.

### Programming Details

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

Working inward from the largest component, the GUI shown in Figure 2 is structured as follows.

- The outermost `JFrame` contains a single component, `CrossWord`, derived from `Box` (in `javax.swing`).
- The `CrossWord`, is a horizontal `Box` that contains a `CrossWordGrid`, and two `WordColumns`. Between the adjacent child objects are two struts (created with `Box.createHorizontalStrut`). The struts ensure at least a little space between the components.
- The `WordColumns` are subclassed from `JPanel` and have layout `BoxLayout` (subclassing from `Box` might have been easier). Each `WordColumn` consists of a `JLabel` and a `WordList`.
- Each `WordList` is subclassed from `JTextArea`. The `TextArea` is set up with

  ```
  this.setEditable(false);
  this.setFont(getWordListFont());
  this.setAlignmentX(LEFT_ALIGNMENT);
  ```

  where `getWordListFont()` returns a `java.awt.Font` object for 14 point Courier.

  The content of the `TextArea` is set with a combination of `String.format` and `TextArea.append`. `this.setColumns` is explicitly set.
- We return to describing the `CrossWordGrid`. The `CrossWordGrid` is a `JPanel` whose `Layout` manager is explicity set to `null`. It has $r \times c$ child objects, one for each square.

  It explicitly sets its own size to $(40r + 6, 40c + 6)$. *If you use a container without a layout manager, be sure to set the size of both the container and its children.*

  The `CrossWordGrid` overrides `paintComponent` to paint its own background dark gray. This colour shows through between the `GridSquares`.
- The child objects of the `CrossWordGrid` are `GridSquares`. Each child object sets its own size to $40 \times 40$, and also sets its location (that is of its top left-corner location relative to the top-left corner of the parent `CrossWordGrid`).

The grid in the first component is the hardest component to display. The approach described 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) ;
```

The individual cells of the can be subclassed from `JComponent`, in which case, you want to override the

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

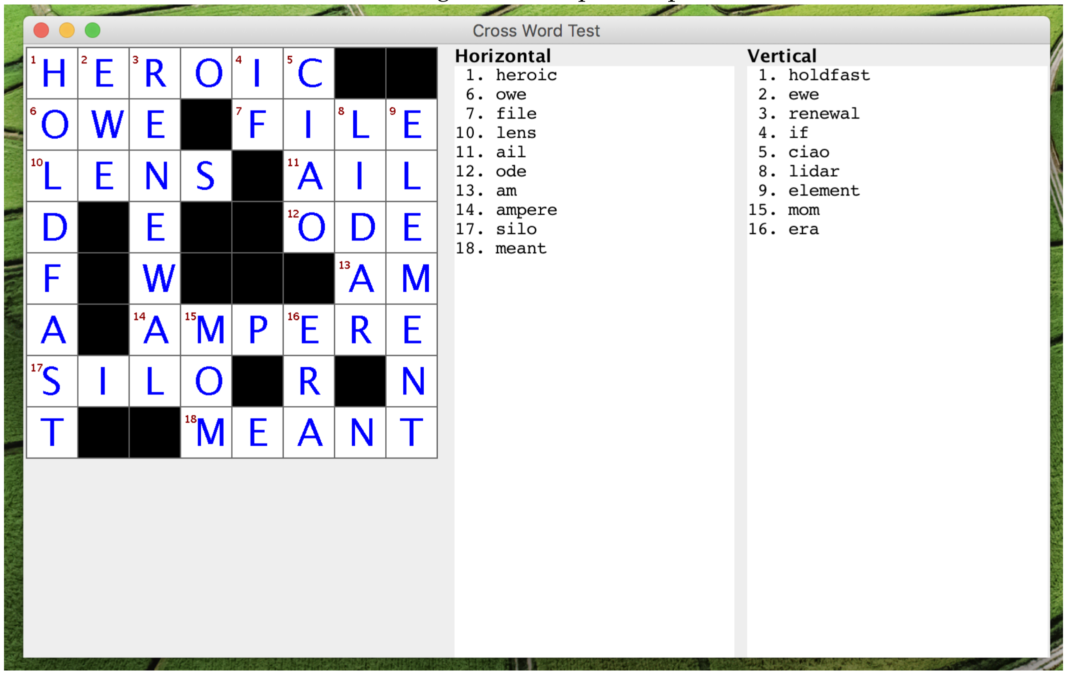
method in order to the paint the cell. Remember to call `super.paintComponent(g)` first, and then do whatever painting you need. You probably want to do a `g.setColor` one or more times, a `g.setFont` zero or more times, and then various `g.fillRect`'s and `g.drawStrings`. The graphics object that you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

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

Figure 1: Sample Input

```
in.txt
8
8
heroicXX
oweXfile
lensXail
dXeXXode
fXwXXXam
aXampere
siloXrXn
tXXmeant
```

Figure 2: Sample Output



```
Horizontal
 1. heroic
 6. owe
 7. file
10. lens
11. ail
12. ode
13. am
14. ampere
17. silo
18. meant
```

```
Vertical
 1. holdfast
 2. ewe
 3. renewal
 4. if
 5. ciao
 8. lidar
 9. element
15. mom
16. era
```

```
FontMetrics fm = graphics.getFontMetrics(getFont()) ;
... fm.charWidth(myChar);
... fm.getStringBounds(""+myChar,0,1,arg0).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.

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 10.13.3 . Other OS's will look slightly different.

Using the fonts and sizes as above, the two-digit square numbers actually collide with wide characters ('`M`'s and '`W`'s). To avoid this, wider characters have been shifted slightly right. This looks slightly silly (check out the '`W`'s in Figure 2). Doubtless, you will find a more elegant solution.