6.005 Elements of Software Construction
Fall 2008

Optional Lab: Graphical User Interfaces in Swing

6.005 Elements of Software Construction
Fall 2008
No due date

In this lab, you will become familiar with GUI programming and the Java Swing user interface toolkit. You will learn about:

- Swing widgets, including windows, labels, text fields, lists, scroll panes, menu bars, and buttons;
- using a layout manager to automatically arrange widgets in a window;
- using action listeners to respond to user input;
- and using a standard dialog provided by Swing.

Swing, and graphical user interface programming in general, is filled with complex APIs and complicated control flow mechanisms. This lab should give you the basic tools you need to complete your GUI in Project 3.

> **Note:** this lab is not required, but you may find it useful.

Before Lab

Before starting this lab, please do the following:

- Read both Project 3 and this lab handout.
- Check out the `guiwords` project from your personal SVN repository.
- If you are new to GUI or Swing programming, familiarize yourself with The Swing Tutorial. Don't try to read the entire tutorial, but do have an idea of what you can find there. Here are some of the most useful sections:
  - Using Swing Components, which includes a visual index of available components and lots of how-tos, is the most immediately useful. In this lab, you will already be using a top-level container, buttons, text fields, labels, lists and models, scroll panes, menus, dialogs, and others.
  - Writing Event Listeners discusses just that.
  - Concurrency in Swing covers some important details of multithreaded GUI programming.

- Read through the tutorial on TableLayout.

Word Finder

Word Finder will be a simple application that presents a basic but functional interface for searching a list of words.

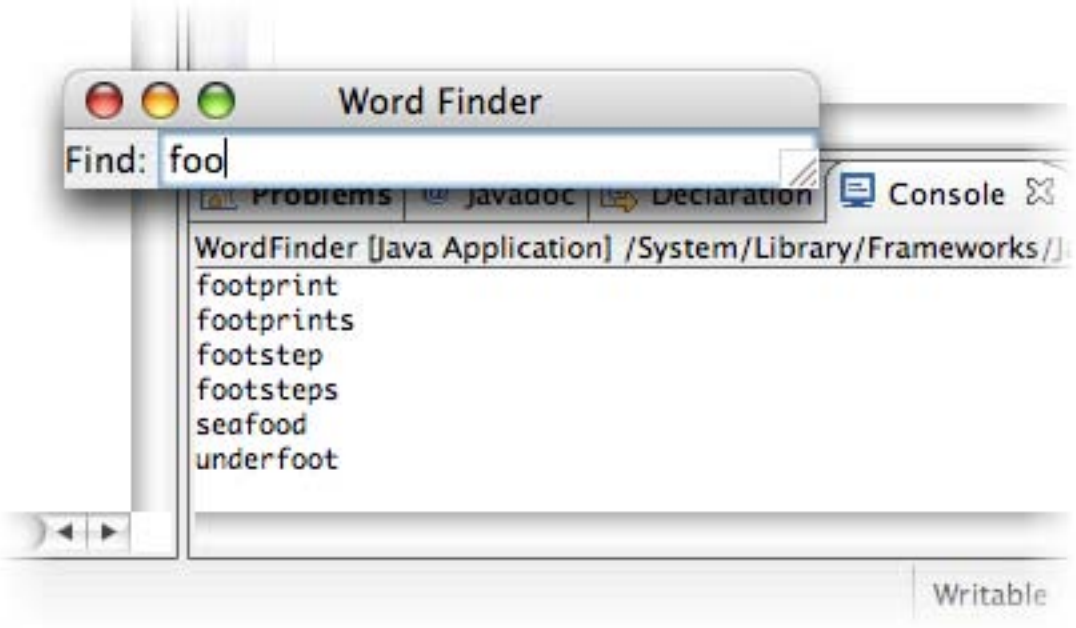
The skeleton of this application is provided for you as follows:

- `words`: a dictionary of about 45,400 words taken from the standard Linux `/usr/share/dict/words`.
- `WordList`: a backend class that represents a list of words and provides operations for loading the list from a stream and searching the list.
- `WordFinder`: a skeleton for the user interface you will implement in this lab.

Getting Started

Begin by running the main methods in `WordList` — it should output a list of words containing "ph" — and `WordFinder`, which should display a very preliminary UI.

Task 1: Add an ActionListener to the `find` JTextField so that pressing "enter" uses `words` to search for the current text in the field and display the results on the console (i.e., `System.out`).



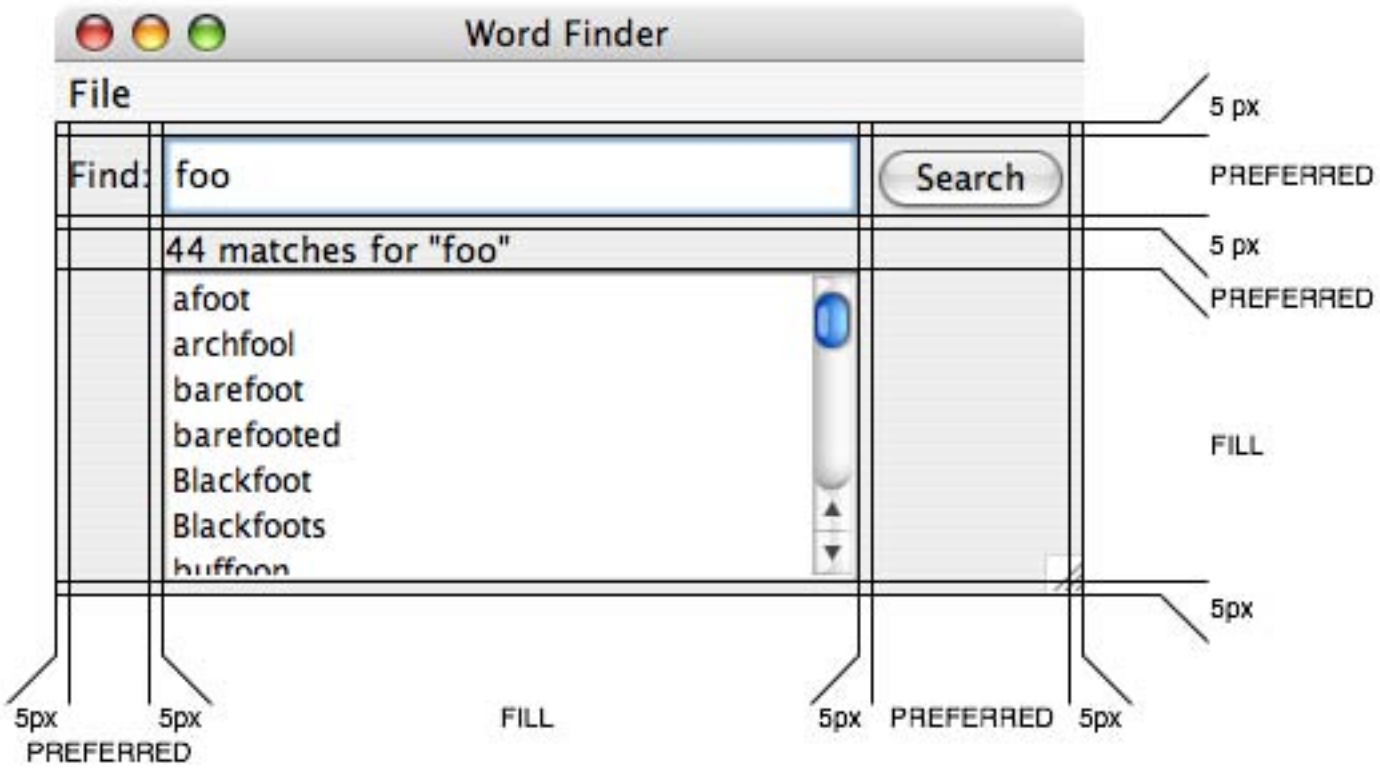
Layout, Lists, and Labels

A LayoutManger determines how components within a container like JFrame are arranged. The default `LayoutManger` for `JFrames`, BorderLayout, is simple but not very powerful. Java provides several other layout managers, some of which are notoriously complicated.

TableLayout is the alternative we will use for this lab, and which you are encouraged to use for the project. Read the documentation and this tutorial.

TableLayout arranges the user interface in a logical table of cells. A 2D array of doubles is used to specify the percentage or absolute width and height of the columns and rows of the table.

We would like to lay out the final interface of the Word Finder application like this:



In this diagram, PREFERRED indicates that the width or height of the column or row is determined by the "preferred" size of the components in it, and FILL indicates that the column or row expands to take up any remaining space when the window is shown or its size is changed by the user.

Task 2: Set the `JFrame`'s layout manager to an appropriately-initialized `TableLayout`. Update the `add(...)` component calls to replace the `BorderLayout` information with the `"column, row"` `String` used by `TableLayout`.

A JList is the appropriate component to display the list of matched words. A `JList` separates the presentation of those words from the list itself by having a separate ListModel. For this lab, you can safely use a DefaultListModel, and add or remove items from this model as needed.

Task 3: Add to the window a `JList` contained inside a JScrollPane. Modify your code so that instead of outputting the matched words to the console, they appear in the scrollable list.

`JScrollPane` provides scrolling behavior for components that are too large to display in their entirety and is one example of how the *view hierarchy* (`JList` inside `JScrollPane` inside `JFrame`) is used to control component display.

Task 4: In addition to the list of results, it is also useful to know the number of matched words. Add a JLabel to your interface that is updated after every search to display the number of matches.

Click Me!

It is essential to organize your implementation so that actions are separated from the particular GUI components that trigger them.

Task 5: Add a "Search" button to the interface so that in addition to pressing "enter," the user can click "search" to update the list of matched words. If necessary, refactor your implementation so that code is not duplicated. At the same time, you should also ensure that when the interface is first displayed, it is identical to what appears when the user searches for the empty string.

Finally, no interface would be complete without a good old fashioned menu bar. And no user interface toolkit would be complete without providing standard dialogs for actions that ought to be consistent across applications, such as choosing a file from the disk. We'll put both of those features to use.

Task 6: Give your interface a JMenuBar with a "File" JMenu. On this menu, have at least two options:

- **Open...**, which should use a JFileChooser to show an "Open File" dialog in which the user can choose a new word list. Note that `WordList` already provides a `load(...)` method once you have obtained an `InputStream` for the chosen file.
- **Exit**, which should... exit the application, for example by using System.exit(...).

The Javadoc documentation for `JFileChooser` includes an example that should make this task straightforward. JOptionPane is another important source for standard dialog boxes.

Finishing Touch

Task 7: You choose...

To finish off this simple Word Finder interface, implement one more feature of your choosing, as time in the lab permits. This can be anything you like; here are some suggestions:

- Add a "Clear" button that clears the text field and resets the current search.
- Add a check box to enable or disable case-sensitivity in the search.
- Use a different listener on the text field so that searches happen *incrementally* as you type, and pressing "enter" or clicking "search" is unnecessary.
- Change the list display so that the part of each word that matches the user's search term is highlighted. (Hint: Swing supports basic HTML, which you may find helpful.)
- Improve the performance of Word Finder and eliminate the word list loading delay by implementing a custom list model.