**KV4001 Week 11 Lab**

**GUI Lab**

(Based on material from MIT)

# The task

In this lab you are going to build a small user interface that searches a list of words. We provide a backend class that does the heavy lifting (actually loading the word list and searching it). We also specify the design of the user interface. Your job is to implement it using the Java Swing user interface toolkit.

To do this assignment, you'll need to know the following:

how to program in Java: writing, compiling, and running Java programs ;



how to use Swing widgets: windows, labels, text fields, buttons, text areas and scroll panes;

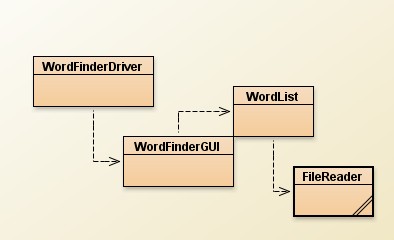
how to use a layout manager to lay out widgets in a window automatically;



how to use listeners to respond to user input;

# Provided Resources

Your application will consist of 4 classes:



We provide you with the following:

 WordList.java: a backend class that represents a list of words and provides operations for loading it from a file and searching it.

FileReader.java: to read in the word set from a file.



WordFinderDriver.java: the application main method.

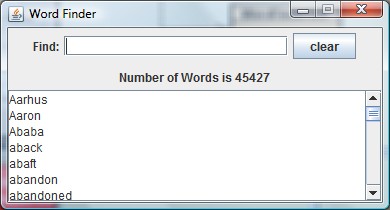
WordFinderGUI.java: A partially completed GUI.

[words](http://groups.csail.mit.edu/graphics/classes/6.831/handouts/ps0/words) : a dictionary of around 45000 words taken from the standard Linux

/usr/share/dict/words.

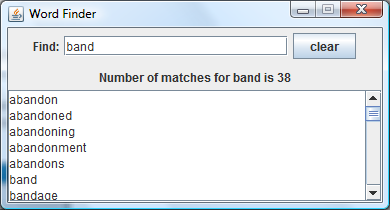
# Problem

You should build an interface that looks like this:

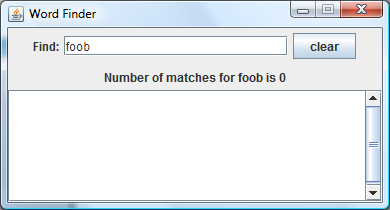


The interface should have the same layout as shown above. When the program is first run, the list box should display the word list we gave you.

The Find text field is the user's query. When the query is blank, the list box displays the entire word list, as shown above. Whenever a query is entered the text area updates to display all words that contain the query text:



If none of the words contain the query, the list box should be empty:



The Clear button should clear the query field, restoring the list box to displaying all words again.

The window should not be resizable.

Clicking the X icon should close the program.

# Task 1

Produce a plan showing how you will design the GUI. This should be a picture of the GUI showing the

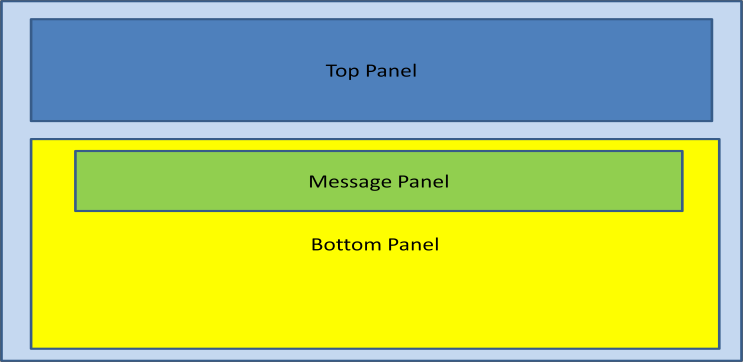
layout manager you have adopted for the containers you propose plus their required layout managers.

# Task 2

You have been given a starter copy of the file WordFinderGUI.java. Your task is to supply the missing

code.

As a starting point you should check to see if your GUI plan agrees with the one proposed in this file:



Add the code for:

makeTopPanel() makeBottomPanel() makeMessagePanel() makeWordsArea() addActionLIsteners()



Once you have done this compile and run the GUI to ensure it appears as required

# Task 3

Complete the actionPerformed method. You are given the steps required for each event.

# Observations

When you run the application you will find it takes a **very long time** to start up and to respond to the

clear button. This is because of the way we are placing the words for display in the text area. The append() method is string based and most probably involves String concatenation. String concatenation is very time consuming. Because we are dealing with over 45000 words we see the slowness.

If you need to concatenate strings , especially when you are dealing with large numbers, you should use

the StringBuilder class. This is far more efficient when it comes to String concatenation.

# Task 4

Replace displayAllWords() and displayMatches() with:

private void displayAllWords()

{

ArrayList<String> allWords = words.getAllWords();

resultMessage.setText("Number of Words is " + allWords.size() );

wordArea.setText(buildString( allWords));

}

private void displayMatches(String charSequence)

{

ArrayList<String> matches = words.find(charSequence);

resultMessage.setText("Number of matches for " + charSequence + " is " + matches.size() );

wordArea.setText(buildString( matches));

}

Below are two versions of buildString(Array<String> list).

Page **4** of **5**

// Version 1

private String buildString( ArrayList<String> list)

{

String temp = "";

for ( String word : list ) temp += word + "\n";

return temp;

}

// Version 2

private String buildString( ArrayList<String> list)

{

StringBuilder temp = new StringBuilder(); for ( String word : list )

temp.append( word + "\n");

return temp.toString();

}

Version 1 uses String concatenation. Add this to the GUI. Compile and run the application. Version 2 uses a StringBuilder. Replace version 1 in your code with version 2. Compile and run the application.

Do you notice any difference in performance? What have you learnt?