Louie Nicholas Lee
Language Helper

A1: The problem

Introduction

Learning a new language is time-consuming. Students have to start out from the basics, memorizing a plethora of basic words (such as small, medium, and big), before they begin learning how to read, write and speak. For students, just memorizing these basic words is not enough. In their courses, they must delve into increasingly more complicated topics, which require specific jargon. Without the proper vocabulary students will find themselves lagging behind the class. Thus memorization is key in learning a new language.

The intended **end-user** is the head of the language department at ISM (Mr. Pethan), with the intent on using it on middle school students. These students are just beginning their studies in a new foreign language and require a basic set of vocabulary before studying the language in greater depth, which is why they require a means of memorizing.

When teaching languages Mr Pethan needs:

- To pass out the vocabulary set to students
    - Must be so that they can refer to it when reviewing for future reference
- To get students to memorize vocabulary sets
- To test that the students have memorized the vocabulary set
- To keep the students engaged

However the current system that they implement has limitations, as described below.

Current System

There are many different ways that students currently try to memorize words:

- Pictorial association
- Read, cover, check
- Rote repetition

Each method has its drawbacks. To illustrate these methods, I visited the class and observed the current system.

(1) Pictorial Association:

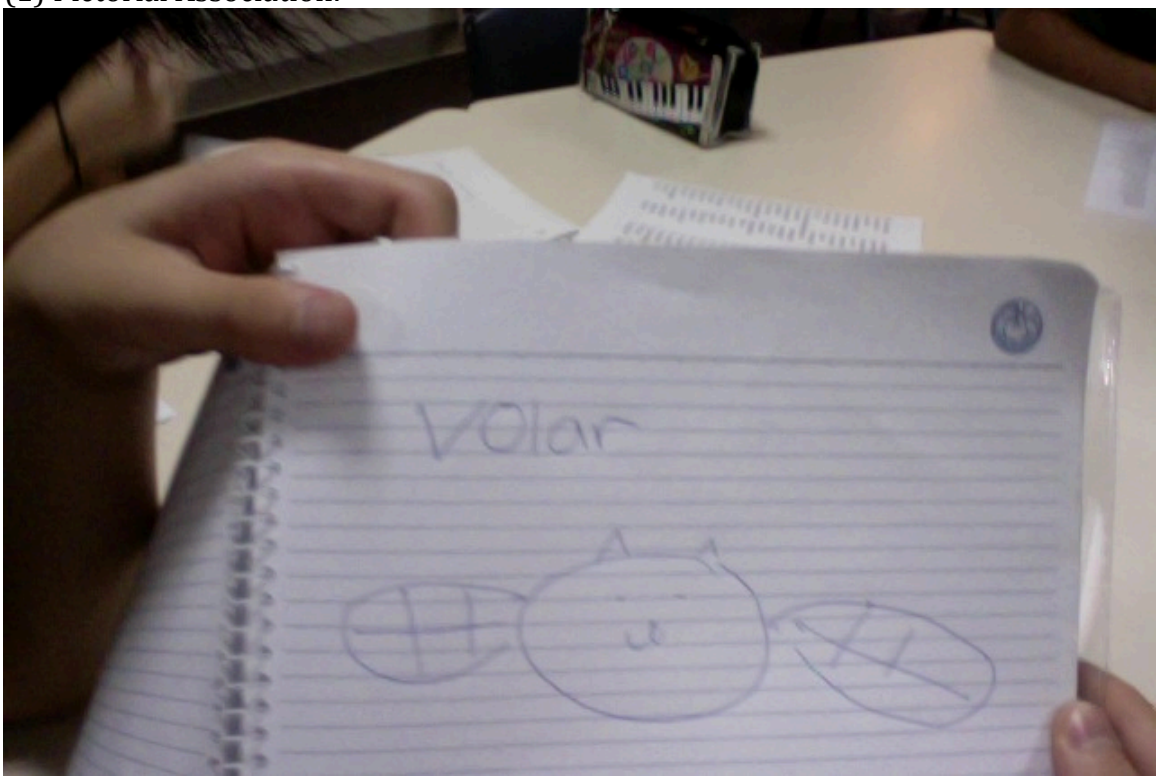


Image 2: Student's drawing of the verb "to fly".

The image above illustrates the way a student might try to memorize a word by associating it with a graphic representation of it. Mr. Pethan has said that it is one of the better methods of memorizing because when students engage their sense of sight they are able to remember things more easily

Disadvantages:

- Given a batch of words, a student has to draw a pictorial representation of each word. This would be very time consuming.
- This also means that for each student, he will have to use up a lot of paper.

(2) Read, cover, check:

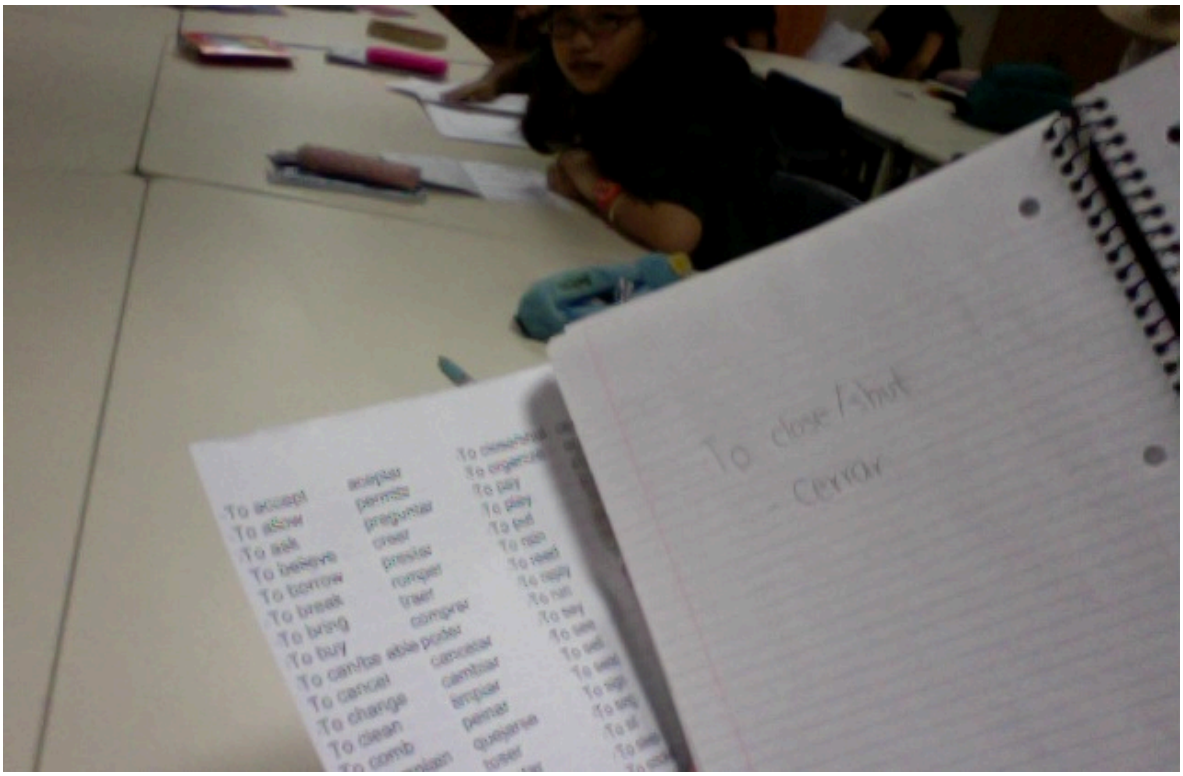


Image 3: Student trying to recollect the covered portion of the word list—the translated version of the word

Another method that Mr. Pethan's students use is brute force memorization. The advantage of this is that students are able to memorize the words quickly.

Disadvantage:

- There is no context of how to use the word (under what situation the word should be used); the student only knows what the words mean.

This is very similar to the third method that Mr. Pethan's students use, rote repetition.

(3) Rote Repetition:

Image 4: A student's attempt at rewriting the words in the list they were given

After students have memorized the words in class, Mr. Pethan assigns a test to the class to measure their progress. This simply makes sure that they know the word and the meaning.

Problems with Current System:

- When students are unsure of a word, they have to ask Mr. Pethan. This takes time away from the other things that he has to do, such as prepare course materials and other important things.

- When students finish the test, Mr. Pethan has to manually check each student's paper. This is cumbersome and repetitive, taking precious time out of his schedule.
- Another big problem is that there are students who regularly lose their vocabulary sheets and have to get another copy when they are reviewing a specific set of vocabulary for a test.

A2: Criteria for Success

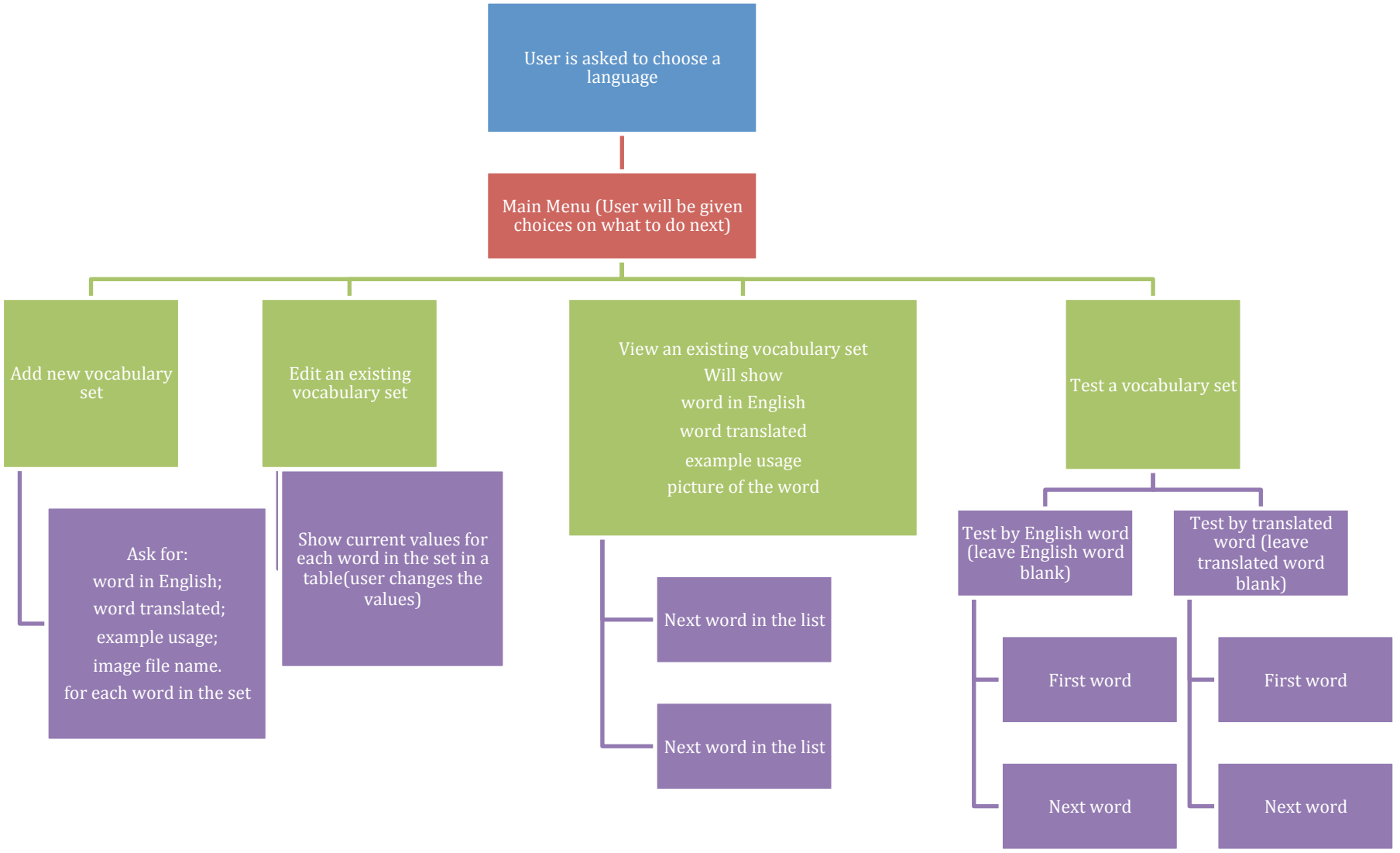
To create a better solution for the teachers, the goals below were set.

Goals	**Problem Addressed**	**Limitation**
The program must be self-explanatory—the functions of the program must be laid out so the user knows what each function does without asking questions.	Because Mr. Pethan already has a lot of work to do, having to explain to his student how to use the program would increase his workload.	The labels for the functions must be very detailed, which could crowd the screen. A help menu will be beneficial if the functions become complicated.
The user should be able to add a picture to each word he/she saves.	Instead of having each student drawing a pictorial representation of the word, wasting lots of paper, the program will do it once and save it for reference for the students to use.	The images may take a lot of space. The program should limit each picture to 500 KB.
The user should be able to add words in a set that are stored for future use.	This will make sure that the students who are prone to losing their vocabulary list can access it when they use the program.	This will also take up a lot of memory.
The words should include example usage of the words, which the students could edit themselves as they see fit.	This will make sure that the students are able to understand the context of the word. They have no need to bother Mr. Pethan because they can already see how the word is used.	N/A

The students should be able to test his self to see whether he has memorized the words and give feedback without the need for the teacher to verify that they are correct.	This will save the trouble of having to create a quiz and taking time out of class to assess the student's knowledge	N/A
The students should be able to track the progress of the words he memorized (should save the words the user hasn't memorized, based on the test, in a new list of words).	This will ensure that the students memorize all the words correctly.	N/A

A3: Prototype Solution

Initial Design



Prototype

When the user starts the program, he will be prompted to choose a language that he/she wants to study and work with. The three choices are Chinese, French and Spanish.

Image 5: Start-up screen

User Feedback:
It's simple and good for our purposes. Clear and concise.

After choosing a language, they will be directed to the main window, which shows the current available word lists. If the user presses the wrong language or wants to try another language to study, they can press the change language button and be redirected to the previous window.

The four main functions of the main window are: adding a new word list, editing an existing word list, viewing an existing word list, and testing their knowledge on a word list.

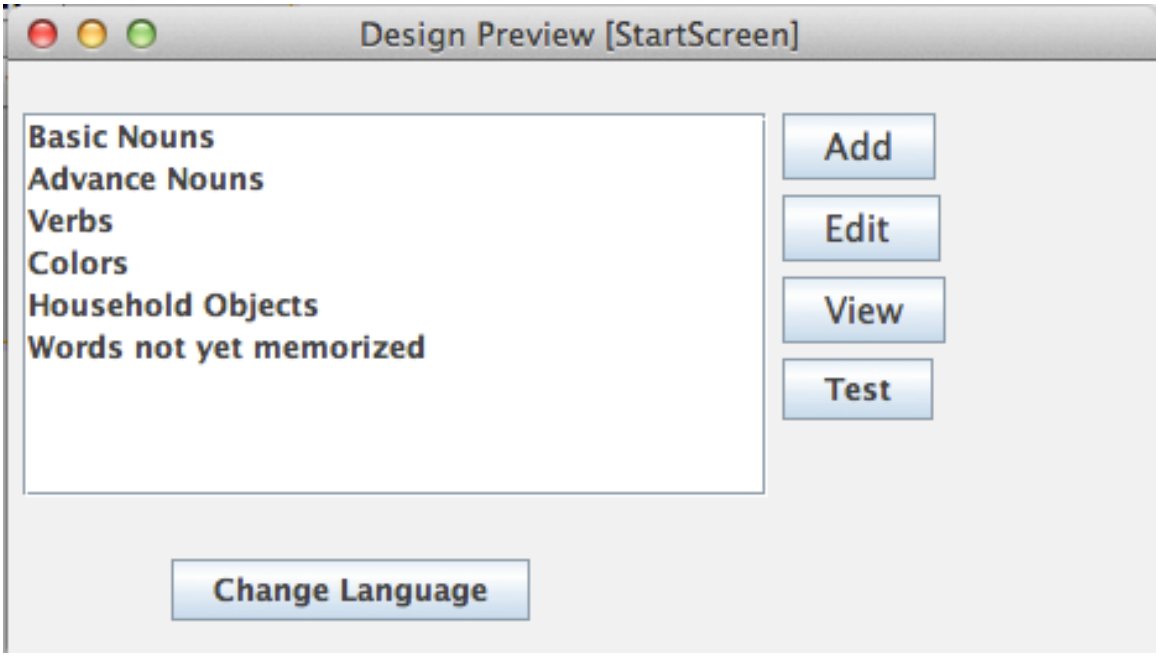


Image 6: Main window

User Feedback:
Clear functions. Simple and effective. Not too complicated so that it is self explanatory for the students to use.

If the user clicks add or edit, the AddMenu window will appear, as shown below. This window allows the user to enter the words they wish to remember. The entries include the word in English, the word in the translated language, a sample usage of the word, and an image title that links to an image that expresses the word. An additional feature in this window is the ability to sort these words in a particular order, which will be used to display the words.

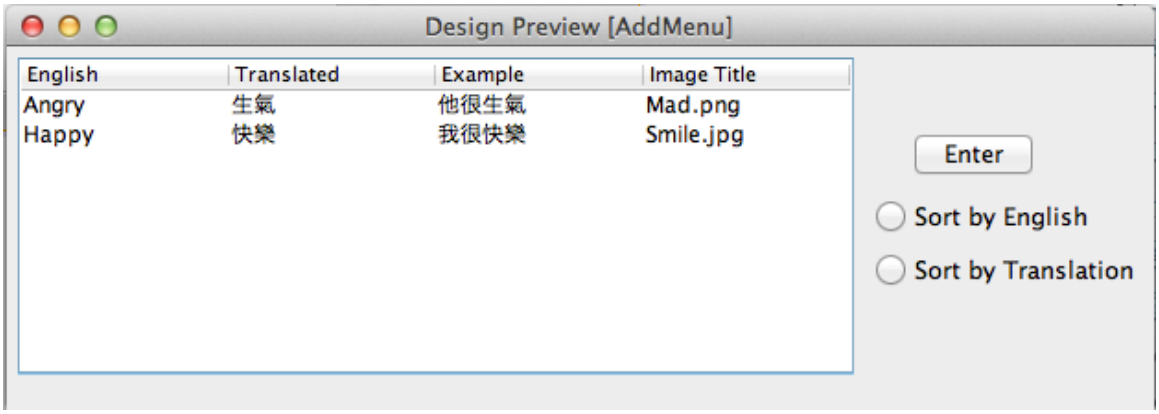


Image 7: AddMenu window

User Feedback:
Perhaps add pronunciation aspect in the program.

If the user clicks the view button, he will be directed to the view window where the picture, the English and translated version, and example usage of the word will be shown. The order in which these words appear will be in the same order as they were entered, or in the order that was selected when the list was created. After clicking next, the user will be shown another word, in an identical window. When the user presses next on the last word, a message displaying "The list is done" will appear and the user will be sent back to the main window.

Image 8: ViewWords menu

User Feedback:
Photo is shown to increase the memorization. Engages the senses, which help in memorizing words. Picture is of appropriate size.

If the user chooses the test function, he will be directed to the window below. Here he will be asked to fill out one of the fields, either English or translated version of the word. If he enters the correct answer he will move on to the next word in the list. If, however, he enters the wrong answer, a message will appear telling him to try again. Three incorrect answers and the program will display a message showing the correct answer and then the user will move on to the next word. After finishing the whole list, the user will be shown the number of correct words he answered. The incorrect words will be saved in another list called "words not yet memorized" for future references.

Image 9: TestWords window

User Feedback:
Having the spelling aspect is important to ensuring the student's knowledge on the subject.

Additional changes:
Mr. Pethan's feedback to add a pronunciation field so that the students are able to know how to pronounce a word will be included. However, because in French and Spanish, there is no regulated way of representing how to pronounce a word, this will be an optional field that only appears if the user has entered it, if not it will be blank.

B1: Data Structures

To get a clear idea of how the program should work—how each function relates to each other—the specific data structures required to create this solution must be discussed in detail. The major components of this program are:

1. Binary Trees
2. Word Array
3. Data Files
    a. RandomAccessFiles
        i. The titles of the vocabulary set will be written via RandomAccessFiles.
    b. Hashtag based delimiter
        i. Each vocabulary set (ie the words) will be written to its own file with the file name being "vocabulary set name.txt"
4. Classes:
    a. Word—an object used to save the words that the user enters
        i. englishWord; String; the English version of the word
        ii. translatedWord; String; the translated version of the word

iii. example; String; an example usage of the translated word in a sentence
iv. pronunciation; String; pronunciation of the word (Optional as shown below in Inheritance and Polymorphism section)
v. imgName; String; the name of the image used that is accessed by the program.

b. Node—used to create the binary search tree
i. next; Node; a reference to the next node in the list
ii. leftChild/rightChild; Node; a reference to the left and right subtree
iii. wordSet; String; the txt file holding the words in the set

Arrays (wordArray):

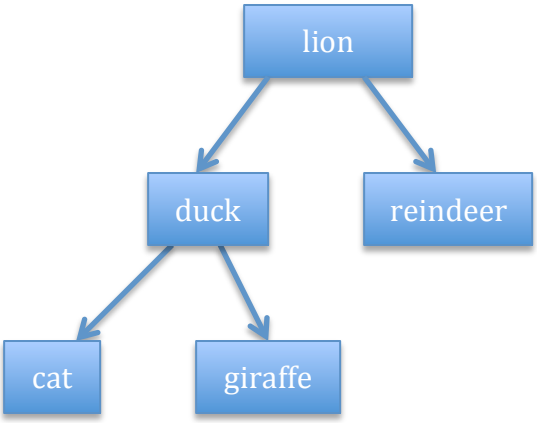
An array of words will be used to temporarily store the words in a vocabulary set. There will be a set number of words per vocabulary set (20) because of the memory issues discussed in section A1/2. This static data structure makes it easy to store the data. There is no need to delete from the array as this is only used for temporary data handling such as display the words. Direct manipulation of the file will be used to delete the arrays. As such, the use of an array is as efficient as other dynamic data structures.

This data structure will look like:

Word[0]	Word[1]	Word[2]	Word[3]
String englishWord String translatedWord String example String pronunciation String imgName	String englishWord String translatedWord String example String pronunciation String imgName	String englishWord String translatedWord String example String pronunciation String imgName	String englishWord String translatedWord String example String pronunciation String imgName

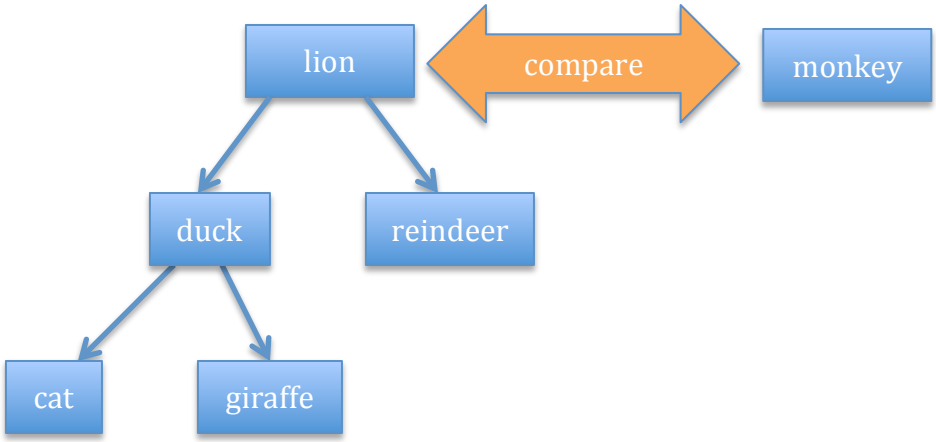
Binary Trees (wordTree):

The wordTree will host the words that have not yet been memorized by the user. This means that from all the different vocabulary sets, the wordTree will have at least one element (word). This means in time the wordTree could become very big. This requires an efficient way to work with the methods: add and remove. The user will memorize these words in the future, so appropriate remove methods must be implemented.

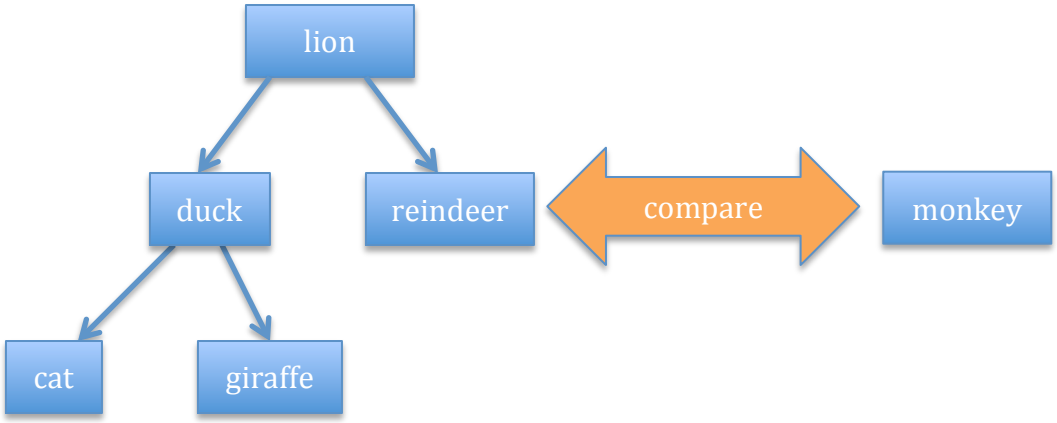


Each subsequent word is added by making comparison and if the word should come before it will compare with the left subtree, if the word should come after it will compare with the right subtree. This comparison is made until the end there is no left subtree remaining and the word is added.

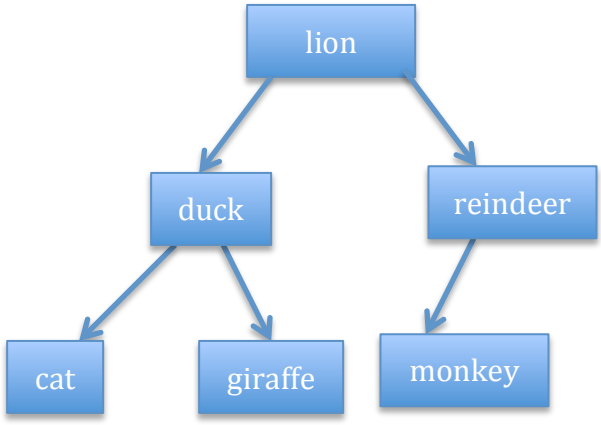
If the animal monkey is added, it will first compare with lion.



Since monkey comes alphabetically after lion, it will compare with the right subtree, reindeer.



A comparison is made; since it comes before reindeer it is compared with the left subtree. Since there are no more words that come to the left of reindeer, monkey is added to the left subtree.



A binary tree is used because it has the most efficient add and delete methods and can easily be used to certain words in the list.

Storage in Disk

Storage of Words:
The data in a specific vocabulary set will be saved in a txt file, which will work on the basis of a asterisk based delimiter—an asterisk in particular because commas and regular punctuation may be used in the example usage for each word. Each vocabulary set will be saved in a different txt file as mentioned before. The program will be able to read these files by accessing the titles RandomAccessFile.

English	Translated	Pronunciation	Example	Image Name
Monkey	猴子	Hóuzi	伊朗上月 28 日宣布把一只活體猴子成功送入地球次軌道，□打算	Monkey.jpg

			2020 年前實現載人航天	
Giraffe	長頸鹿	Chángjǐnglù	另外，瑞典飛機旅館、肯亞的長頸鹿飯店，都靠著新奇的創意讓住房天天客滿	Giraffe.jpg
Lion	獅子	Shīzi	競爭並不是壞事，跑得過獅子的羚羊得以繁衍	Lion.jpg

When saved on disk the text file will look like:

Monkey*猴子*Hóuzi*伊朗上月 28 日宣布把一只活體猴子成功送入地球次軌道，□打算 2020 年前實現載人航天*Monkey.jpg
Giraffe*長頸鹿*Chángjǐnglù*另外，瑞典飛機旅館、肯亞的長頸鹿飯店，都靠著新奇的創意讓住房天天客滿*Giraffe.jpg
Lion*獅子*Shīzi*競爭並不是壞事，跑得過獅子的羚羊得以繁衍*Lion.jpg

An asterisk delimiter separated file is used because the length of the string fields of each word is variable.

Storage of vocabulary set titles:
It is important that the program is able to track the list of word sets that has been saved, therefore a txt file (Word Sets Titles.txt) will be created solely including the names of the txt files of the different word sets. Unlike the variable lengths of the string fields in Word, the titles will be set length of 50 and will be divided equally.

This will look like:
Monkey Nouns Adjective

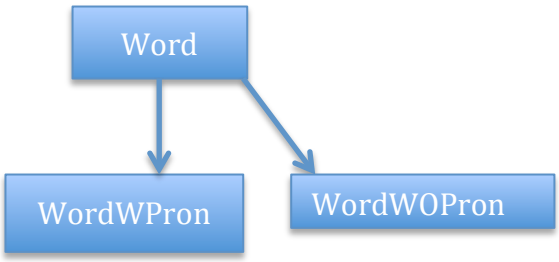
With 50 characters between each word set title.

RandomAccessFiles is used to directly manipulate certain titles that have been edited without having the need to change the whole file or reading sequentially.

NOTE: These files will be saved in directories based on their languages: Chinese directory, French directory, Spanish directory. They will be independent of each directory.

Inheritance and Polymorphism

An important part of the program is to be able to sort the words in different order. While it may seem simple, the way of ordering is different in certain languages. A language can be a phonetic based language like Chinese, which has pronunciation, or it can be alphabet based like French and Spanish with no set notation on the pronunciation. Therefore there will be two types of words WordWPron and WordWOPron, which will extend the Word object.



French and Spanish will be identical to each other, with Chinese being different. There will be an overloaded method of adding the words to the binary tree, additional features pronunciation for the Chinese language.

B2: Algorithms

<table>
<tr><td colspan="3">Algorithm Name: addWordtoTree</td></tr>
<tr><td>Description: adds a word to the binary search tree of word set. This algorithm will run once the word has been created. This algorithm is separate from the addWord because it will be used later to add the incorrect words that are answered into a new set of words.</td><td>Parameters passed:
word (Object type Word); node (Object type Node)</td><td>Parameters Returned:
None</td></tr>
<tr><td colspan="2" rowspan="3">Pseudo Code:
<pre>
if node = null
     node = new Node (word)
else if node.word comes before word
     if node.left = null
          node.left = new Node (word)
     else addWordtoTree (node.left,
word)
ekse if node.word comes after word
     if node.right = null
          node.right = new Node (word)
     else addWordtoTree (node.right,
word)
</pre></td><td>Outputs:
None</td></tr>
<tr><td>Precondition: the word must not be null; node must be referencing the root of binary tree</td></tr>
<tr><td>Postcondition: The word is added to the correct place in the binary search tree</td></tr>
</table>

<table>
<tr><td colspan="3">Algorithm Name: addWord</td></tr>
<tr><td>Description: adds a word to the binary search tree of word set.</td><td>Parameters passed:
N/A</td><td>Parameters Returned:
N/A</td></tr>
<tr><td colspan="2" rowspan="3">Pseudo Code:
<pre>
Create an instance of word of object
Word
     Set englishWord = get english word
from user
     Set translatedWord = get
translated word from   user
     Set pronunciation = get
pronunciation from user
     Set example = get example from
user
     Set imgName = get imgName from
</pre></td><td>Outputs:
None</td></tr>
<tr><td>Precondition: There must be inputs from the user</td></tr>
<tr><td>Postcondition: places the word in the binary search tree. Adds the new word to the file containing the word set.</td></tr>
</table>

user addWordtoTree(head, word) write word to file	

Algorithm Name: editWord		
Description: edits a word in the binary search tree of word set	**Parameters passed:** word	**Parameters Returned:** N/A
Pseudo Code: Set englishWord = get english word from user Set translatedWord = get translated word from user Set pronunciation = get pronunciation from user Set example = get example from user Set imgName = get imgName from user Search for word in file and delete word Write word to file		**Outputs:** None **Precondition:** The word must not be null (must have been initialized) **Postcondition:** changes the word to what the user wants. Deletes the old word and adds the new word to the file containing the word set.

Algorithm Name: delete		
Description: deletes a word in the binary search tree	**Parameters passed:** Word (word to be deleted) node (a node of the binary tree that contains word)	**Parameters Returned:** N/A

Pseudo Code:
if node is null do nothing
if word is before node
delete(word, node.leftSubTree)
if word is after node
delete (word, node.rightSubTree)
else
if node.leftSubTree and node.rightSubTree = NULL
node = NULL
if(node.leftSubTree = NULL)
tmp = node
node = node.rightSubTree
if(node.rightSubTree = NULL)
tmp = node
node = node.leftSubTree
		Outputs: None
		Precondition: None
		Postcondition: deletes word

```
     else
          tmp = the next closest word
in right            subtree
          Set node's word = tmp's word
          delete (tmp's word,
node.rightSubTree)
```

Algorithm Name: setQuestions		
Description: creates the questions that will test the user whether he has memorized the word or not	**Parameters passed:** word (Object type Word) int (Type int will indicate which type of test will be used)	**Parameters Returned:** None

Pseudo Code:

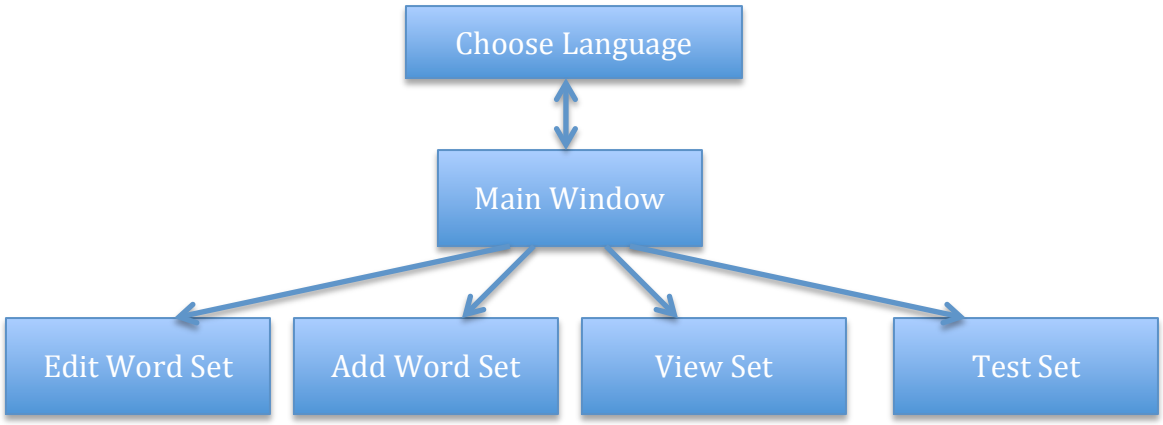
```
if int == 1
     show English word in text field
     show Example in text field
     if pronunciation is empty
          show blank locked text field
     show blank text field for
else
     search correct placement of word*
     insert word
```

Outputs:
None

Precondition: the word must not be null

Postcondition: The word is added to the correct place in the binary search tree

B3: Modular Organization



Choose Language:
This portion will allow the user to choose which language he wants to use the program for.

Called from	Links to
Main Window	Main Window

The data structure that will be used here is the different languages, Chinese, Spanish and French.

Main Window:
Shows the list of sets of words that have been created by the user. Contains the main functions of the program where the user can manipulate the chosen set of words.

Called from	Links to
Choose Language	Choose Language Edit Word Set Add Word Set View Set Test Set

The data structure that will be used in this is the linked list of word sets, which will be taken from the fileNames.txt
Main algorithm used: Load fileNames.

Edit Word Set:
Allows the user to edit the pre-existing word sets in the program. After editing, they will be redirected to show the created set of words.

Called from	Links to
Main Window	View Set

The data structure that will be used is the wordTree. This requires the specific txt file of the word set.

Main algorithm used: editWord, delete.

Add Word Set:
The function that creates new word sets. After creating the word sets, the user will be redirected to the view set module where they can view the created word set.

Called from	Links to
Main Window	View set

Data structures that will be used are wordTree and linked list of word sets. This will write to a new txt file and requires the fileNames.txt.
Main algorithm used: addWord, addWordToTree.

View Set:
The user is prompted with the words that are created, showing the picture, English word, translated word, example and pronunciation. There will be several windows of different words in the set.

Called from	Links to
Main Window Edit Word Set Add Word Set Test Set	Test Set Main Window View Set

Data structure that will be used is wordTree. This will require the specific txt file for the word set and the image file.
Main method: traversal.

Test Set:
The user will be tested on the list he has chosen. It will show some blanks that the user will have to fill in. Each question, once answered, will link to the next question. Once the user finishes the test, they will be directed to the Main Window if they pass, if not they are redirected to view set.

Called from	Links to
Main Window View Set	Test Set Main Window View Set

Data structure that will be used is wordTree. It will also use the scores data type. This will require the specific txt file for the word set.
Main Method: checkAnswer, createQuestion, addWordtoTree.

Mastery Index

1. **Encapsulation**
   See Word class.
   All instance variables are private to restrict user-allowed actions to getting and setting via get, set methods
2. **Inheritance**
   See Word class, WordWOPron class, WordWPron class.
   The WordWOPron and WordWPron classes inherit the Word class. This is because this program works with different languages, both with pronunciation (character based languages such as Chinese) and without (alphabetic based languages). As such, an inheritance of the member variables is required, with the word with pronunciation class adding the Pronunciation member variable.
3. **Polymorphism**
   See AddWord form.
   There exist two constructors that allow the AddWord form to act as both a way to edit the words and to add the words: done by passing the argument of a Word in one of the constructors. This is a non-trivial use of polymorphism because otherwise there would have to be two forms created.
4. **Parsing a text file or other data stream**
   See WordTree class.
   The populate method reads text files to load the words as a Word object to the Binary search tree. Also readTitles method in AppLogic class allows the user to read saved vocabulary set titles from previous runs of the program.
5. **Adding data to an instance of the RandomAccessFile class by direct manipulation of the file pointer using the seek method**
   See AppLogic class.
   The writeTitle and editTitle method uses RAF to add the titles in the proper place using seek. This is important because it allows for direct writing of the titles in correct place without the need for searching the file. editTitle takes the index of the title that needs to be edited.
6. **Deleting data from an instance of the RandomAccessFile class by direct manipulation of the file pointer using the seek method**
   See AppLogic class.
   The deleteTitle method uses RAF to find the word it needs to delete using the seek method to track which word it is in. It uses an input index and uses that to seek directly to the portion of the file that the title that needs to be deleted is in, no extra manipulation of data is required.
7. **Recursion**

   See WordBinTree class.
   The add, delete, find methods in the WordBinTree is recursive. Alternative non-recursive method would be much more cumbersome.

8. **The use of any five standard level mastery factors—this can be applied only once**
    a. **Arrays**
    See the different frames, i.e. EditSetForm.
    This uses an array of Words to populate the jList.
    b. **User-defined objects**
    See Word class.
    Word is a user-defined object
    c. **Complex selection**
    See AddWordForm.
    This uses complex selection to inform the user of the different errors that could occur when trying to save a word.
    d. **Loops**
    See AppLogic class.
    readTitles method uses for loop to read the characters in from file
    numWordsSet method uses while loop to count number of lines in the file to return the number of words in the set
    e. **User-defined methods**
    See AppLogic class.
    Delete method is a user-method.
9. **ADT (Binary Tree 4 Aspects)**

    See WordTree class.
    There are proper constructors and get, set methods. The add and delete methods are implemented. And proper error handling is used to ensure that the user does not enter the same word twice, unless it has different traits (if so the overwritten compareTo method will be used).
10. **Use of additional libraries (such as utilities and graphical libraries not included in appendix 2 Java Examination Tool Subsets)**
    The GUI of this program uses swing components such as the JOptionPane, which shows a message to the user (used in AddWordsForm/ViewForm).
    Another example is the use of Arrays utility to pass only the populated values in the array to the WordSetForm List this is important so as not to populate the values in the list with null objects. (see EditSetForm.java, EditActionPerformed)
11. **Implementing a hierarchial data Structure**
    wordTree class is a binary tree that holds records of Word.
    There are several instances of Word[] arrays being used throughout the program such as in AppLogic.
    These data structures make it easy to work with the data.

14/10 Mastery factors

C1: Using good programming style

Main.java

```
 1 /*
 2  * To change this template, choose Tools | Templates
 3  * and open the template in the editor.
 4  */
 5
 6 /**
 7  * @author "Louie Nicholas Lee, June 2012, International School Manila"
 8  * Program: Language Helper
 9  * Computer: 13-inch MacBook Pro, IDE: NetBeans
10  * Purpose: To help facilitate learning of new vocabulary in foreign languages
11  * Starts the program
12  */
13
14 public class Main {
15     public static void main (String[] args){
16         AppLogic al = new AppLogic();
17         StartScreenForm start = new StartScreenForm(al);
18         start.setVisible(true);
19     }
20 }
21
```

Word.java

```
 1 /**
 2  *
 3  * @author louienicholaslee, June 2012, International School Manila 13-inch
 4  * MacBook Pro, NetBeans. Purpose: To help facilitate learning of new vocabulary
 5  * in foreign language.
 6  *
 7  * Object record used to store the foreign language word, meaning, example usage
 8  * and image path name Word is an abstract class because a word cannot be just a
 9  * word, it must either be a word with pronunciation or a word without
10  * pronunciation this is important because the program works with words
11  * character based languages which require pronuciation and alphabet based
12  * languages which don't.
13  *
14  * Mastery Factor: User-defined objects
15  */
16 public abstract class Word {
17     //stores the meaning of the word
18
19     private String word;
20     //stores the word
21     private String meaning;
22     //stores example usage of the word in a sentence
23     private String example;
24     //stores the name of the image path that illustrates the meaning of the word
25     private String imgName;
26
27 //-------------------------------Accessor Methods-----------------------------//
28     //Mastery Factor: Encapsulation
```

```
29  public String getWord() {
30      return word;
31  }
32
33  public void setWord(String s) {
34      word = s;
35  }
36
37  public String getMeaning() {
38      return meaning;
39  }
40
41  public void setMeaning(String s) {
42      meaning = s;
43  }
44
45  public String getExample() {
46      return example;
47  }
48
49  public void setExample(String s) {
50      example = s;
51  }
52
53  public String getImgName() {
54      return imgName;
55  }
56
57  public void setImgName(String s) {
58      imgName = s;
```

```
59   }
60 //---------------------------------------------------------------------------//
61
62   //Overrides the compareTo method to compare the different fields of the word
63   //This is done because of homonym words.
64   public int compareTo(Word w) {
65      int before = -1;
66      int equal = 0;
67      int after = 1;
68
69      if (this.word.compareToIgnoreCase(w.word) < 0) {
70         return before;
71      } else if (this.word.compareToIgnoreCase(w.word) > 0) {
72         return after;
73      } else {
74         if (this.meaning.compareToIgnoreCase(w.meaning) < 0) {
75            return before;
76         } else if (this.meaning.compareToIgnoreCase(w.meaning) > 0) {
77            return after;
78         } else {
79            if (this.imgName.compareToIgnoreCase(w.imgName) < 0) {
80               return before;
81            } else if (this.imgName.compareToIgnoreCase(w.imgName) > 0) {
82               return after;
83            } else {
84               return equal;
85            }
86         }
87      }
88   }
```

```
89
90    //Overrides the toString method that objects have. Used to display the Word
91    //object.
92    public String toString() {
93       return word + "#" + meaning + "#" + example + "#" + imgName;
94    }
95 }
96
97 //Mastery Factor: Inheritance
98 class WordWPron extends Word {
99    //stores the pronunciation of the word
100
101    private String pronunciation;
102
103    //Constructor for the WordWPron automatically sets the member fields using
104    //arguments
105    WordWPron(String word, String meaning, String ex, String img, String pron) {
106       setWord(word);
107       setMeaning(meaning);
108       setExample(ex);
109       setImgName(img);
110       setPronunciation(pron);
111    }
112
113 //-------------------------------Accessor Methods-----------------------------//
114    public String getPronunciation() {
115       return pronunciation;
116    }
117
118    public void setPronunciation(String s) {
```

```
119       pronunciation = s;
120    }
121 //-------------------------------------------------------------------------//
122
123    //Mastery Factor: Polymorphism
124    //Overrides compareTo method in abstract class adding another comparisson with
125    //the pronunciation of the word
126    public int compareTo(Word w) {
127       int before = -1;
128       int equal = 0;
129       int after = 1;
130       if (super.compareTo(w) == 0) {
131          if (getPronunciation().compareToIgnoreCase(((WordWPron) w).getPronunciation()) < 0) {
132             return before;
133          } else if (getPronunciation().compareToIgnoreCase(((WordWPron) w).getPronunciation()) > 0) {
134             return after;
135          } else {
136             return equal;
137          }
138       } else {
139          return super.compareTo(w);
140       }
141    }
142
143    //Mastery Factor: Polymorphism
144    //Overrides toString method in abstract class appending the pronunciation
145    public String toString() {
146       return super.toString() + "#" + getPronunciation();
147    }
148 }
```

```
149
150 //Mastery Factor: Inheritance
151 class WordWOPron extends Word {
152
153    //Constructor for the WordWOPron automatically sets the member fields using
154    //arguments
155    WordWOPron(String word, String meaning, String ex, String img) {
156       setWord(word);
157       setMeaning(meaning);
158       setExample(ex);
159       setImgName(img);
160    }
161
162    //Mastery Factor: Polymorphism
163    public int compareTo(WordWOPron w) {
164       return super.compareTo(w);
165    }
166
167    //Mastery Factor: Polymorphism
168    public String toString() {
169       return super.toString();
170    }
171 }
172
```

WordTree.java

```
 1
 2 import java.io.BufferedReader;
 3 import java.io.BufferedWriter;
 4 import java.io.FileNotFoundException;
 5 import java.io.FileOutputStream;
 6 import java.io.FileReader;
 7 import java.io.IOException;
 8 import java.io.OutputStreamWriter;
 9 import java.io.UnsupportedEncodingException;
10
11 /**
12  *
13  * @author louienicholaslee, June 2012, International School Manila 13-inch
14  * MacBook Pro, NetBeans. Purpose: To help facilitate learning of new vocabulary
15  * in foreign languages
16  *
17  * BinarySearchTree used to store temporarily the words not yet memorized.
18  * Mastery Factor: ADT Tree
19  */
20 public class WordTree {
21
22     private NodeBinTree root = null;
23
24     //accessor method (root cannot be changed by outside classes unless via the
25     //clear method)
26     public NodeBinTree getRoot() {
27         return root;
28     }
```

```
29
30  //checks to see if the WordTree is empty
31  public boolean isEmpty() {
32     if (root == null) {
33        return true;
34     } else {
35        return false;
36     }
37  }
38
39  //add method with the appropriate check to make sure that if the tree is empty
40  //it adds to the root. Also checks for duplicate adding. Primarily used to
41  //populate the binary tree
42  public void add(Word w) {
43     if (root == null) {
44        root = new NodeBinTree(w);
45     } else {
46        if (!find(w)) {
47           root.add(w);
48        }
49     }
50  }
51
52  //add method with the appropriate check to make sure that if the tree is empty
53  //it adds to the root. Also checks for duplicate adding. Primarily used to
54  //add words that have not yet been memorized to the word tree and directly
55  //writes the words to file
56  public void insert(Word w) {
57     if (root == null) {
58        root = new NodeBinTree(w);
```

```
59      } else {
60          if (!find(w)) {
61              root.add(w);
62          }
63      }
64      //writes the word to file
65      BufferedWriter out = null;
66      try {
67          try {
68              out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(
69                      AppLogic.getWorkingDirectory() + "/Words Not Yet Memorized.txt", true), "UTF-8"));
70          } catch (UnsupportedEncodingException ex) {
71              ex.printStackTrace();
72          }
73          try {
74              out.write(w.toString() + "\n");
75              out.close();
76          } catch (IOException ex) {
77              ex.printStackTrace();
78          }
79      } catch (FileNotFoundException ex) {
80          ex.printStackTrace();
81      }
82  }
83
84  //removes the word from the binary tree
85  public void remove(Word w) {
86      if(!isEmpty()){
87      root = root.remove(w);
88      }
```

```
89  }
90
91  //clears the binary tree so that the whole tree can be rewritten again
92  public void clear() {
93     root = null;
94  }
95
96  //populates the binary tree from the appropriate language using the Working
97  //directory
98  //Mastery Factor: Parsing a file
99  public void populate(String workingDir) {
100     boolean phoneticLanguage = false;
101     if (workingDir.equalsIgnoreCase("Chinese")) {
102        phoneticLanguage = true;
103     } else {
104        phoneticLanguage = false;
105     }
106     BufferedReader in;
107     try {
108        in = new BufferedReader(new FileReader(workingDir + "/Words Not Yet Memorized.txt"));
109        String line;
110        String word;
111        String meaning;
112        String example;
113        String imgName;
114        String pronunciation;
115        String[] wordComponents = null;
116        while (true) {
117           line = in.readLine();
118           if (line == null) {
```

```
119             break;
120          }
121          wordComponents = line.split("#");
122          word = wordComponents[0];
123          meaning = wordComponents[1];
124          example = wordComponents[2];
125          imgName = wordComponents[3];
126          if (phoneticLanguage) {
127             pronunciation = wordComponents[4];
128             add(new WordWPron(word, meaning, example, imgName, pronunciation));
129          } else {
130             add(new WordWOPron(word, meaning, example, imgName));
131          }
132       }
133       in.close();
134    } catch (IOException ex) {
135       ex.printStackTrace();
136    }
137 }
138 
139 //searches whether the word is in the binary tree or not. Returs true if found
140 //false if not
141 public boolean find(Word w) {
142    if (isEmpty()) {
143       return false;
144    } else {
145       if (root.find(w) != null) {
146          return true;
147       } else {
148          return false;
```

```
149         }
150       }
151    }
152
153    //saves the word tree to memory
154    public void saveWordTree(){
155       root.saveWordTree();
156    }
157 }
158
```

NodeBinTree.java

```
 1
 2 import java.io.BufferedWriter;
 3 import java.io.FileNotFoundException;
 4 import java.io.FileOutputStream;
 5 import java.io.IOException;
 6 import java.io.OutputStreamWriter;
 7 import java.io.UnsupportedEncodingException;
 8
 9 /*
10  * To change this template, choose Tools | Templates
11  * and open the template in the editor.
12  */
13
14 /**
15  *
16  * @author louienicholaslee, June 2012, International School Manila 13-inch
17  * MacBook Pro, NetBeans. Purpose: To help facilitate learning of new vocabulary in
18  * foreign languages
19  *
20  * NodeBinTree-a Binary Search Tree Node
21  * Mastery Factor: ADT Tree
22  */
23 public class NodeBinTree {
24     //stores the word as a key
25     private Word word;
26     private NodeBinTree left;
27     private NodeBinTree right;
28
```

```
29    //constructor for a NodBinTree class
30    NodeBinTree(Word w) {
31       word = w;
32       right = null;
33       left = null;
34    }
35
36 //------------------------------Accessor Methods-----------------------------//
37    public Word getWord() {
38       return word;
39    }
40
41    public void setWord(Word w) {
42       word = w;
43    }
44
45    public NodeBinTree getLeft() {
46       return left;
47    }
48
49    public NodeBinTree getRight() {
50       return right;
51    }
52
53    public void setLeft(NodeBinTree l) {
54       left = l;
55    }
56
57    public void setRight(NodeBinTree r) {
58       right = r;
```

```
59    }
60 //-------------------------------------------------------------------------//
61
62    //Mastery Factor: Recursion
63    //Recursive add function which takes in a word parameter
64    public void add(Word w) {
65       if (w.compareTo(word) < 0) {
66          if (left == null) {
67             left = new NodeBinTree(w);
68          } else {
69             left.add(w);
70          }
71       } else {
72          if (right == null) {
73             right = new NodeBinTree(w);
74          } else {
75             right.add(w);
76          }
77       }
78    }
79
80    //Mastery Factor: Recursion
81    //Recursive search function used to see whether a word has been memorized yet
82    //or not. Returns null if the word memorized
83    public Word find(Word w) {
84       Word result = null;
85       if (w.compareTo(word) == 0) {
86          result = word;
87       } else {
88          if (w.compareTo(word) < 0) {
```

```
89              if (left != null) {
90                  result = left.find(w);
91              }
92          } else {
93              if (right != null) {
94                  result = right.find(w);
95              }
96          }
97      }
98      return result;
99  }
100
101  //Mastery Factor: Recursion
102  //Recursive delete function, deletes the word and corrects the tree. Returns
103  //the pointer to the reconstructed portion of the tree
104  public NodeBinTree remove(Word w){
105      NodeBinTree result = this;
106      if (w.compareTo(word) == 0) {
107          if (left == null && right == null) {
108              result = null;
109          } else if (left != null && right == null) {
110              result = left;
111          } else if (left == null && right != null) {
112              result = right;
113          } else {
114              //getSuccessor is used to find the closest word that comes after
115              //the current word.
116              result = getSuccessor();
117              result.left = left;
118              result.right = right;
```

```
119         }
120      } else {
121         if (w.compareTo(word) < 0) {
122            if (left != null) {
123               left = left.remove(w);
124            }
125         } else {
126            if (right != null) {
127               right = right.remove(w);
128            }
129         }
130      }
131      return result;
132   }
133
134   //Mastery Factor: Recursion
135   //Recursive functions which searches for the next closest word that comes after
136   //the current word. Also removes the word that was found and returns it.
137   public NodeBinTree getSuccessor() {
138      NodeBinTree successor = right;
139      while (successor.left != null) {
140         successor = successor.left;
141      }
142      right.remove(successor.word);
143      return successor;
144   }
145
146   //saves the word tree in memory via pre-order
147   public void saveWordTree(){
148      BufferedWriter out = null;
```

```
149       try {
150          try {
151             out = new BufferedWriter(new OutputStreamWriter(
152                   new FileOutputStream("Words Not Yet Memorized.txt"), "UTF-8"));
153          } catch (UnsupportedEncodingException ex) {
154             ex.printStackTrace();
155          }
156          try {
157             out.write(this.word.toString());
158             if(left!=null){
159                left.saveWordTree();
160             }
161             if(right!= null){
162                right.saveWordTree();
163             }
164             out.close();
165          } catch (IOException ex) {
166             ex.printStackTrace();
167          }
168       } catch (FileNotFoundException ex) {
169          ex.printStackTrace();
170       }
171    }
172 }
173
```

AppLogic.java

```
 1
 2 import java.io.BufferedReader;
 3 import java.io.BufferedWriter;
 4 import java.io.EOFException;
 5 import java.io.File;
 6 import java.io.FileNotFoundException;
 7 import java.io.FileOutputStream;
 8 import java.io.FileReader;
 9 import java.io.IOException;
10 import java.io.OutputStreamWriter;
11 import java.io.RandomAccessFile;
12 import java.io.UnsupportedEncodingException;
13
14 /**
15  *
16  * @author louienicholaslee, June 2012, International School Manila 13-inch
17  * MacBook Pro, NetBeans. Purpose: To help facilitate learning of new vocabulary in
18  * foreign languages
19  *
20  * AppLogic class controls the logic of between GUI, program logic, and files stored
21  * in RAM
22  */
23 public class AppLogic {
24     //A static member variable that is constant throughout all instances of
25     //the AppLogic. Stores the number of bytes that a title (title of vocabulary
26     //sets) record can have
27     public static final int TITLE_RECORD_SIZE = 50;
28     //Dictates the maximum number of words that can be in a vocabulary set
```

```
29    public static final int MAX_NUM_WORDS = 20;
30    //A static wordTree that stores the words that have not yet been memorized
31    //as a binary search tree.
32    public static WordTree wordsNotMemorized = new WordTree();
33    //Boolean showing if the current language being worked on is phonetic (ie has
34    //pronunciation
35    private boolean phoneticLanguage;
36    //Shows the current working directory: Chinese, French or Spanish. Changes
37    //depending on the language being worked on
38    private static String workingDir;
39
40 //-------------------------------Accessor Methods-----------------------------//
41    public static String getWorkingDirectory() {
42       return workingDir;
43    }
44
45    public void setWorkingDirectory(String dir) {
46       workingDir = dir;
47       if (workingDir.equalsIgnoreCase("chinese")) {
48          phoneticLanguage = true;
49       } else {
50          phoneticLanguage = false;
51       }
52       //creates the Words Not Yet Memorized file if not already done so
53       File file = new File(workingDir + "/Words Not Yet Memorized.txt");
54       try {
55          file.createNewFile();
56       } catch (IOException ex) {
57          ex.printStackTrace();
58       }
```

```
59      //clears the current wordTree
60      wordsNotMemorized.clear();
61      //populates wordTree with words not yet memorized from the chosen language
62      wordsNotMemorized.populate(workingDir);
63    }
64
65    public boolean isPhoneticLanguage() {
66      return phoneticLanguage;
67    }
68  //-------------------------------------------------------------------------//
69
70    //Mastery Factor: Deleting RandomAccessFile using seek method
71    //Deletes a title from the saved titles file by direct manipulation of file,
72    //using the index number of the title that will be deleted
73    public void deleteTitle(int index) {
74      try {
75        RandomAccessFile raf = new RandomAccessFile(workingDir + "/Word Sets Titles.txt", "rw");
76        int numTitlesToMove = (int) (raf.length() - (index + 1) * TITLE_RECORD_SIZE) / TITLE_RECORD_SIZE;
77        int posTitleToDelete = index * TITLE_RECORD_SIZE;
78        long newRAFLength = raf.length() - TITLE_RECORD_SIZE;
79        if (numTitlesToMove != 0) {
80          raf.seek(posTitleToDelete + TITLE_RECORD_SIZE);
81          String titlesToRewrite = raf.readLine();
82          raf.seek(posTitleToDelete);
83          raf.writeBytes(titlesToRewrite);
84          raf.setLength(newRAFLength);
85        } else {
86          raf.setLength(newRAFLength);
87        }
88        raf.close();
```

```
89      } catch (IOException ex) {
90          ex.printStackTrace();
91      }
92  }
93
94  //Edits the file directly via RandomAccessFile with parameters of index and
95  //title that will be changed to
96  //Mastery Factor: User-defined methods
97  public void editTitle(int index, String title) {
98      try {
99          RandomAccessFile raf = new RandomAccessFile(workingDir + "/Word Sets Titles.txt", "rw");
100         int posTitleToEdit = index * TITLE_RECORD_SIZE;
101         raf.seek(posTitleToEdit);
102         raf.writeBytes(title);
103         for (int i = 0; i < (TITLE_RECORD_SIZE - title.length()); i++) {
104             raf.writeBytes(" ");
105         }
106         raf.close();
107     } catch (IOException ex) {
108         ex.printStackTrace();
109     }
110 }
111
112 //Mastery Factor: Adding RandomAccessFile using seek method
113 //Adds the tile title to the file
114 public void writeTitle(String title) {
115     try {
116         RandomAccessFile raf = new RandomAccessFile(workingDir + "/Word Sets Titles.txt", "rw");
117         try {
118             if (raf.length() == 0) {
```

```
119             raf.writeBytes(title);
120          } else {
121             raf.seek(raf.length() + ((TITLE_RECORD_SIZE - raf.length()) % TITLE_RECORD_SIZE));
122             raf.writeBytes(title);
123          }
124          for (int i = 0; i < (TITLE_RECORD_SIZE - title.length()); i++) {
125             raf.writeBytes(" ");
126          }
127          raf.close();
128       } catch (IOException ex) {
129          ex.printStackTrace();
130       }
131    } catch (FileNotFoundException ex) {
132       ex.printStackTrace();
133    }
134 }
135
136 //Reads the titles from the file and returns a string array of the titles
137 //Mastery Factor: loops
138 public String[] readTitles() {
139    String[] titles = null;
140    try {
141       RandomAccessFile raf = new RandomAccessFile(workingDir + "/Word Sets Titles.txt", "rw");
142       int numTitles = (int) (raf.length() / TITLE_RECORD_SIZE);
143       if (wordsNotMemorized.isEmpty()) {
144          titles = new String[numTitles];
145       } else {
146          titles = new String[numTitles + 1];
147          titles[numTitles] = "Words Not Yet Memorized";
148       }
```

```
149         for (int i = 0; i < numTitles; i++) {
150             byte[] titleCharArray = new byte[TITLE_RECORD_SIZE];
151             try {
152                 for (int j = 0; j < TITLE_RECORD_SIZE; j++) {
153                     titleCharArray[j] = raf.readByte();
154                 }
155             } catch (EOFException e) {
156                 e.printStackTrace();
157             }
158             titles[i] = new String(titleCharArray).trim();
159         }
160         raf.close();
161     } catch (IOException ex) {
162         ex.printStackTrace();
163     }
164     return titles;
165 }
166 
167 //Writes/rewrites the whole vocabulary set to the proper txt file using title
168 //parameter and wordArray
169 
170 public void saveWordSet(Word[] wordArray, String title) {
171     BufferedWriter out = null;
172     try {
173         int fileLength = wordArray.length;
174         try {
175             //first empties the file so that when written, the is no leftover
176             //entries that were written before
177             deleteSet(title);
178             //UTF-8 allows writing of chinese and accent based characters
```

```
179             out = new BufferedWriter(new OutputStreamWriter(
180                     new FileOutputStream(workingDir + "/" + title + ".txt"), "UTF-8"));
181         } catch (UnsupportedEncodingException ex) {
182             ex.printStackTrace();
183         }
184         try {
185             for (int i = 0; i < fileLength; i++) {
186                 out.write(wordArray[i].toString() + "\n");
187             }
188             out.close();
189         } catch (IOException ex) {
190             ex.printStackTrace();
191         }
192     } catch (FileNotFoundException ex) {
193         ex.printStackTrace();
194     }
195 }
196 
197 //returns the counts the number of words in the set by counting the number of
198 //lines, returns this as int
199 //Mastery Factor: loop
200 public static int numWordsSet(String title) {
201     BufferedReader reader = null;
202     int lines = 0;
203     try {
204         reader = new BufferedReader(new FileReader(workingDir + "/" + title + ".txt"));
205         while (reader.readLine() != null) {
206             lines++;
207         }
208         reader.close();
```

```
209     } catch (IOException ex) {
210         ex.printStackTrace();
211     }
212     return lines;
213 }
214 
215 //reads the whole vocabulary set from memory using the title parameter, returning
216 //a word Array
217 public Word[] readWordSet(String title) {
218     Word[] wordArray = new Word[numWordsSet(title)];
219     BufferedReader in;
220     try {
221         in = new BufferedReader(new FileReader(workingDir + "/" + title + ".txt"));
222         String line;
223         String word;
224         String meaning;
225         String example;
226         String imgName;
227         String pronunciation;
228         String[] wordComponents = null;
229         int counter = 0;
230         while (true) {
231             line = in.readLine();
232             if (line == null) {
233                 break;
234             }
235             wordComponents = line.split("#");
236             word = wordComponents[0];
237             meaning = wordComponents[1];
238             example = wordComponents[2];
```

```
239             imgName = wordComponents[3];
240             //Distinguishes between phonetic and non-phonetic languages
241             if (phoneticLanguage) {
242                 pronunciation = wordComponents[4];
243                 wordArray[counter++] = new WordWPron(word, meaning, example, imgName, pronunciation);
244             } else {
245                 wordArray[counter++] = new WordWOPron(word, meaning, example, imgName);
246             }
247         }
248         in.close();
249     } catch (IOException ex) {
250         ex.printStackTrace();
251     }
252     return wordArray;
253 }
254
255 //Empties the title.txt file
256 public static void deleteSet(String title) {
257     File file = new File(workingDir + "/" + title + ".txt");
258     file.delete();
259 }
260 }
```

StartScreenForm.java

```
 1
 2 import javax.swing.JOptionPane;
 3
 4 /*
 5  * To change this template, choose Tools | Templates
 6  * and open the template in the editor.
 7  */
 8 /**
 9  *
10  * @author louienicholaslee, June 2012, International School Manila 13-inch
11  * MacBook Pro, NetBeans. Purpose: To help facilitate learning of new vocabulary in
12  * foreign languages
13  *
14  * The main screen that is used throughout. this acts as the main window from which
15  * other forms branch out from
16  */
17 public class StartScreenForm extends javax.swing.JFrame {
18     private AppLogic appLogic;
19
20     /**
21      * Creates new form StartScreen
22      */
23     public StartScreenForm(AppLogic al) {
24         initComponents();
25         this.appLogic = al;
26         //Prompts the user to choose the language they want to learn in
27         ChangeLanguagesForm cl = new ChangeLanguagesForm(this, true);
28         cl.setVisible(true);
```

```
29      //changes the working diretory and language to that chosen by the user
30      appLogic.setWorkingDirectory(cl.getLanguage());
31      //populates the Titles List to show the available vocabulary set
32      Titles.setListData(appLogic.readTitles());
33  }
34
35  /**
36   * This method is called from with6in the constructor to initialize the
37   * form. WARNING: Do NOT modify this code. The content of this method is
38   * always regenerated by the Form Editor.
39   */
40  @SuppressWarnings("unchecked")
41  // <editor-fold defaultstate="collapsed" desc="Generated Code">
42  private void initComponents() {
43
44      jToggleButton1 = new javax.swing.JToggleButton();
45      jScrollPane1 = new javax.swing.JScrollPane();
46      jList1 = new javax.swing.JList();
47      jScrollPane2 = new javax.swing.JScrollPane();
48      jTree1 = new javax.swing.JTree();
49      Test1 = new javax.swing.JButton();
50      ChangeLanguage = new javax.swing.JButton();
51      View = new javax.swing.JButton();
52      Add = new javax.swing.JButton();
53      Edit = new javax.swing.JButton();
54      jScrollPane3 = new javax.swing.JScrollPane();
55      Titles = new javax.swing.JList();
56      Test = new javax.swing.JButton();
57      Delete = new javax.swing.JButton();
58
```

```
59      jToggleButton1.setText("View List");
60
61      jList1.addAncestorListener(new javax.swing.event.AncestorListener() {
62          public void ancestorMoved(javax.swing.event.AncestorEvent evt) {
63          }
64          public void ancestorAdded(javax.swing.event.AncestorEvent evt) {
65              jList1AncestorAdded(evt);
66          }
67          public void ancestorRemoved(javax.swing.event.AncestorEvent evt) {
68          }
69      });
70      jScrollPane1.setViewportView(jList1);
71
72      jScrollPane2.setViewportView(jTree1);
73
74      Test1.setText("Test");
75
76      setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
77
78      ChangeLanguage.setText("Change Language");
79      ChangeLanguage.addActionListener(new java.awt.event.ActionListener() {
80          public void actionPerformed(java.awt.event.ActionEvent evt) {
81              ChangeLanguageActionPerformed(evt);
82          }
83      });
84
85      View.setFont(new java.awt.Font("Lucida Grande", 0, 12)); // NOI18N
86      View.setText("View");
87      View.addActionListener(new java.awt.event.ActionListener() {
88          public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
89            ViewActionPerformed(evt);
90          }
91      });
92
93      Add.setFont(new java.awt.Font("Lucida Grande", 0, 12)); // NOI18N
94      Add.setText("Add ");
95      Add.addActionListener(new java.awt.event.ActionListener() {
96          public void actionPerformed(java.awt.event.ActionEvent evt) {
97              AddActionPerformed(evt);
98          }
99      });
100
101     Edit.setFont(new java.awt.Font("Lucida Grande", 0, 12)); // NOI18N
102     Edit.setText("Edit ");
103     Edit.addActionListener(new java.awt.event.ActionListener() {
104         public void actionPerformed(java.awt.event.ActionEvent evt) {
105             EditActionPerformed(evt);
106         }
107     });
108
109     Titles.setModel(new javax.swing.AbstractListModel() {
110         String[] strings = { "Basic Nouns", "Advanced Nouns", "Verbs", "Colors", "Household Objects", "Words not yet
    memorized" };
111         public int getSize() { return strings.length; }
112         public Object getElementAt(int i) { return strings[i]; }
113     });
114     jScrollPane3.setViewportView(Titles);
115
116     Test.setText("Test");
117     Test.addActionListener(new java.awt.event.ActionListener() {
```

```
118         public void actionPerformed(java.awt.event.ActionEvent evt) {
119             TestActionPerformed(evt);
120         }
121     });
122 
123     Delete.setText("Delete");
124     Delete.addActionListener(new java.awt.event.ActionListener() {
125         public void actionPerformed(java.awt.event.ActionEvent evt) {
126             DeleteActionPerformed(evt);
127         }
128     });
129 
130     org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(getContentPane());
131     getContentPane().setLayout(layout);
132     layout.setHorizontalGroup(
133         layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
134         .add(layout.createSequentialGroup()
135             .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
136                 .add(layout.createSequentialGroup()
137                     .add(12, 12, 12)
138                     .add(jScrollPane3, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 192,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
139                     .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
140                     .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
141                         .add(Add)
142                         .add(Edit)
143                         .add(View)
144                         .add(Test)
145                         .add(org.jdesktop.layout.GroupLayout.TRAILING, Delete)))
146                 .add(layout.createSequentialGroup()
```

```
147                 .add(75, 75, 75)
148                 .add(ChangeLanguage, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 139,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
149             .addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
150       );
151
152       layout.linkSize(new java.awt.Component[] {Add, Delete, Edit, Test, View},
    org.jdesktop.layout.GroupLayout.HORIZONTAL);
153
154       layout.setVerticalGroup(
155           layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
156           .add(layout.createSequentialGroup()
157               .addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
158               .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
159                   .add(layout.createSequentialGroup()
160                       .add(Add)
161                       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
162                       .add(Edit)
163                       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
164                       .add(View)
165                       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
166                       .add(Test)
167                       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
168                       .add(Delete))
169                   .add(jScrollPane3, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 163,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
170               .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
171               .add(ChangeLanguage)
172               .addContainerGap())
173       );
```

```
174
175     pack();
176 }// </editor-fold>
177
178 private void jList1AncestorAdded(javax.swing.event.AncestorEvent evt) {
179     // TODO add your handling code here:
180 }
181
182 //Leads to the add window where the user add a new vocabulary set
183 private void AddActionPerformed(java.awt.event.ActionEvent evt) {
184     //Uses the same form as Edit to simplify code. Difference is that the add
185     //button brings an empty vocabulary set
186     EditSetForm esf = new EditSetForm(this, true, appLogic.isPhoneticLanguage());
187     esf.setVisible(true);
188     if (esf.getTitle() != null) {
189         //if the user has properly added a vocabulary set, the files with the
190         //titles of vocabulary sets is updated and the Titles List is updated
191         //All of the words created is saved to appropriate txt file.
192         appLogic.writeTitle(esf.getTitle());
193         Titles.setListData(appLogic.readTitles());
194         appLogic.saveWordSet(esf.getWordSet(), esf.getTitle());
195     }
196 }
197
198 //Changes the language, prompting the user to choose from chinese, spanish,
199 //french. this edits the working directory of the program
200 private void ChangeLanguageActionPerformed(java.awt.event.ActionEvent evt) {
201     //shows the change language form
202     ChangeLanguagesForm clf = new ChangeLanguagesForm(this, true);
203     clf.setVisible(true);
```

```
204     //changes working directory
205     appLogic.setWorkingDirectory(clf.getLanguage());
206     //updates Titles list with vocabulary set titles from the new language
207     Titles.setListData(appLogic.readTitles());
208   }
209
210   //Similar to add button shows a populated form
211   private void EditActionPerformed(java.awt.event.ActionEvent evt) {
212     //protects from editing a vocabulary set without choosing one
213     if (Titles.getSelectedValue() != null) {
214       //makes sure that words not yet memorized set is not editable
215       if (Titles.getSelectedValue().toString().equalsIgnoreCase("Words Not Yet Memorized")) {
216         JOptionPane.showMessageDialog(null, "Cannot edit this vocabulary set.");
217       } else {
218         //stores the vocabulary set title chosen
219         String editSetTitle = Titles.getSelectedValue().toString();
220         //opens up the new form that is populated with the words already
221         // existing beforehand and the title of the vocabulary set
222         EditSetForm esf = new EditSetForm(this, true, appLogic.readWordSet(editSetTitle),
223               editSetTitle, appLogic.isPhoneticLanguage());
224         esf.setVisible(true);
225         //if user has correctly edited the files and lists are updated
226         if (esf.getTitle() != null) {
227           appLogic.editTitle(Titles.getSelectedIndex(), esf.getTitle());
228           Titles.setListData(appLogic.readTitles());
229           appLogic.deleteSet(esf.getTitle());
230           appLogic.saveWordSet(esf.getWordSet(), esf.getTitle());
231         }
232       }
233     } else {
```

```
234         JOptionPane.showMessageDialog(null, "Please select a vocabulary set to edit");
235       }
236   }
237
238   //Allows the user to delete a vocabulary set
239   private void DeleteActionPerformed(java.awt.event.ActionEvent evt) {
240       //protects from user choosing to delete without having chosen a vocabulary
241       //set to delete
242       if (Titles.getSelectedIndex() != -1) {
243           //protects from user deleting the words not yet memorized set
244           if (Titles.getSelectedValue().toString().equalsIgnoreCase("Words Not Yet Memorized")) {
245               JOptionPane.showMessageDialog(null, "Cannot delete this vocabulary set."
246                       + " You must do the test for this set and answer all questions correctly to delete it");
247           } else {
248               //deletes the vocabulary set title from the file that stores all
249               //the vocabulary set titles
250               appLogic.deleteTitle(Titles.getSelectedIndex());
251               //deletes file that stores the words of the vocabulary set in
252               //memory
253               appLogic.deleteSet(Titles.getSelectedValue().toString());
254               //updates the titles list
255               Titles.setListData(appLogic.readTitles());
256           }
257       } else {
258           JOptionPane.showMessageDialog(null, "Please select a vocabulary set to delete");
259       }
260   }
261
262   //allows the user to view the vocabulary set they wish to see
263   private void ViewActionPerformed(java.awt.event.ActionEvent evt) {
```

```
264     //protects from viewing a non-specified vocabulary set
265     if (Titles.getSelectedIndex() != -1) {
266         //stores the title of the set that will be viewed
267         String viewSetTitle = Titles.getSelectedValue().toString();
268         //opens view form
269         ViewForm vf = new ViewForm(this, true, appLogic.readWordSet(viewSetTitle),
270                 appLogic.isPhoneticLanguage(), Titles.getSelectedValue().toString());
271         vf.setVisible(true);
272     } else {
273         JOptionPane.showMessageDialog(null, "Please select a vocabulary set to view");
274     }
275 }
276 
277 //allows the user to test how well they know the vocabulary set
278 private void TestActionPerformed(java.awt.event.ActionEvent evt) {
279     //protects from going to a test that is not specified
280     if (Titles.getSelectedIndex() != -1) {
281         //stores title of the set that will be tested
282         String testSetTitle = Titles.getSelectedValue().toString();
283         //opens the test form
284         TestForm tf = new TestForm(this, true, appLogic.readWordSet(testSetTitle),
285                 appLogic.isPhoneticLanguage(), Titles.getSelectedValue().toString());
286         tf.setVisible(true);
287         //displays a message if the user has perfected the test
288         if (tf.getNumWrong() > 0) {
289             //if not perfect, a message showing their results is shown
290             JOptionPane.showMessageDialog(null, "You answered " +
291                     (tf.getNumQuestions() - tf.getNumWrong()) +
292                     " questions correctly out of " + tf.getNumQuestions());
293             //the user is then shown the list of words they got incorrect by
```

```
294             //opening the view form
295             ViewForm vf = new ViewForm(this, true, tf.getIncorrectAnswers(), appLogic.isPhoneticLanguage());
296             vf.setVisible(true);
297             Titles.setListData(appLogic.readTitles());
298         } else {
299             JOptionPane.showMessageDialog(null, "Congratulations! You scored perfectly on the test");
300         }
301     } else {
302         JOptionPane.showMessageDialog(null, "Please select a vocabulary set to test yourself with");
303     }
304 }
305 /**
306  * @param args the command line arguments
307  */
308 // Variables declaration - do not modify
309 private javax.swing.JButton Add;
310 private javax.swing.JButton ChangeLanguage;
311 private javax.swing.JButton Delete;
312 private javax.swing.JButton Edit;
313 private javax.swing.JButton Test;
314 private javax.swing.JButton Test1;
315 private javax.swing.JList Titles;
316 private javax.swing.JButton View;
317 private javax.swing.JList jList1;
318 private javax.swing.JScrollPane jScrollPane1;
319 private javax.swing.JScrollPane jScrollPane2;
320 private javax.swing.JScrollPane jScrollPane3;
321 private javax.swing.JToggleButton jToggleButton1;
322 private javax.swing.JTree jTree1;
323 // End of variables declaration
```

```
324 }
325
```

ChangeLanguagesForm.java

```
1 /*
2  * To change this template, choose Tools | Templates
3  * and open the template in the editor.
4  */
5
6 /**
7  *
8  * @author louienicholaslee, June 2012, International School Manila 13-inch
9  * MacBook Pro, NetBeans. Purpose: To help facilitate learning of new vocabulary in
10  * foreign languages
11  *
12  * Allows user to change languages
13  */
14 public class ChangeLanguagesForm extends javax.swing.JDialog {
15
16    /**
17     * Creates new form ChangeLanguages
18     */
19    //stores the language that will be worked on
20    private String language;
21
22    //accesssor method
23    public String getLanguage(){
24        return language;
25    }
26
27    //constructor
28    public ChangeLanguagesForm(java.awt.Frame parent, boolean modal) {
```

```
29      super(parent, modal);
30      initComponents();
31  }
32
33  /**
34   * This method is called from within the constructor to initialize the form.
35   * WARNING: Do NOT modify this code. The content of this method is always
36   * regenerated by the Form Editor.
37   */
38  @SuppressWarnings("unchecked")
39  // <editor-fold defaultstate="collapsed" desc="Generated Code">
40  private void initComponents() {
41
42      Chinese = new javax.swing.JButton();
43      French = new javax.swing.JButton();
44      Spanish = new javax.swing.JButton();
45      jLabel1 = new javax.swing.JLabel();
46
47      setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
48
49      Chinese.setText("Chinese");
50      Chinese.addActionListener(new java.awt.event.ActionListener() {
51          public void actionPerformed(java.awt.event.ActionEvent evt) {
52              ChineseActionPerformed(evt);
53          }
54      });
55
56      French.setText("French");
57      French.addActionListener(new java.awt.event.ActionListener() {
58          public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
59          FrenchActionPerformed(evt);
60       }
61    });
62
63    Spanish.setText("Spanish");
64    Spanish.addActionListener(new java.awt.event.ActionListener() {
65       public void actionPerformed(java.awt.event.ActionEvent evt) {
66          SpanishActionPerformed(evt);
67       }
68    });
69
70    jLabel1.setText("Which language do you want to study?");
71
72    org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(getContentPane());
73    getContentPane().setLayout(layout);
74    layout.setHorizontalGroup(
75       layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
76       .add(layout.createSequentialGroup()
77          .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
78             .add(layout.createSequentialGroup()
79                .add(18, 18, 18)
80                .add(Chinese)
81                .add(18, 18, 18)
82                .add(French)
83                .add(18, 18, 18)
84                .add(Spanish))
85             .add(layout.createSequentialGroup()
86                .add(52, 52, 52)
87                .add(jLabel1)))
88          .addContainerGap(23, Short.MAX_VALUE))
```

```
89      );
90      layout.setVerticalGroup(
91          layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
92          .add(layout.createSequentialGroup()
93              .add(34, 34, 34)
94              .add(jLabel1)
95              .add(18, 18, 18)
96              .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
97                  .add(Chinese)
98                  .add(French)
99                  .add(Spanish))
100             .addContainerGap(31, Short.MAX_VALUE))
101     );
102
103     pack();
104 }// </editor-fold>
105
106 //changes the language to chinese before closing the form
107 private void ChineseActionPerformed(java.awt.event.ActionEvent evt) {
108     language = "Chinese";
109     this.dispose();
110 }
111
112 //changes the language to french before closing the form
113 private void FrenchActionPerformed(java.awt.event.ActionEvent evt) {
114     language = "French";
115     this.dispose();
116
117 }
118
```

```
119    //changes the language to spanish before closing the form
120    private void SpanishActionPerformed(java.awt.event.ActionEvent evt) {
121        language = "Spanish";
122        this.dispose();
123    }
124
125    /**
126     * @param args the command line arguments
127     */
128    // Variables declaration - do not modify
129    private javax.swing.JButton Chinese;
130    private javax.swing.JButton French;
131    private javax.swing.JButton Spanish;
132    private javax.swing.JLabel jLabel1;
133    // End of variables declaration
134 }
135
```

EditSetForm.java

```
 1
 2 import java.util.Arrays;
 3 import javax.swing.JOptionPane;
 4
 5 /*
 6  * To change this template, choose Tools | Templates
 7  * and open the template in the editor.
 8  */
 9 /**
10  *
11  * @author louienicholaslee, June 2012, International School Manila 13-inch
12  * MacBook Pro, NetBeans. Purpose: To help facilitate learning of new vocabulary in
13  * foreign languages
14  *
15  * allows user to edit/add vocabulary sets
16  */
17 public class EditSetForm extends javax.swing.JDialog {
18     //stores the words in the set
19     private Word[] wordArray = new Word[AppLogic.MAX_NUM_WORDS];
20     //stores then number of words in the set
21     private int numWords = 0;
22     private String title = null;
23     private boolean phoneticLanguage;
24
25     /**
26      *
27      * constructor form for editing a preexisting vocabulary set
28      */
```

```
29  public EditSetForm(java.awt.Frame parent, boolean modal, Word[] wordArray, String title, boolean phoneticLanguage) {
30      super(parent, modal);
31      initComponents();
32      this.phoneticLanguage = phoneticLanguage;
33      numWords = wordArray.length;
34      //enters the vocabulary set title and words in the set
35      Title.setText(title);
36      WordSet.setListData(wordArray);
37      for (int i = 0; i < numWords; i++) {
38          this.wordArray[i] = wordArray[i];
39      }
40  }
41
42  //constructor form for adding a new vocabulary set
43  public EditSetForm(java.awt.Frame parent, boolean modal, boolean phoneticLanguage) {
44      super(parent, modal);
45      initComponents();
46      this.phoneticLanguage = phoneticLanguage;
47  }
48
49
50  public Word[] getWordSet() {
51      Word[] finalWordArray = new Word[numWords];
52      for (int i = 0; i < numWords; i++) {
53          finalWordArray[i] = wordArray[i];
54      }
55      return finalWordArray;
56  }
57
58  public String getTitle() {
```

```
59      return title;
60  }
61
62  /**
63   * This method is called from within the constructor to initialize the form.
64   * WARNING: Do NOT modify this code. The content of this method is always
65   * regenerated by the Form Editor.
66   */
67  @SuppressWarnings("unchecked")
68  // <editor-fold defaultstate="collapsed" desc="Generated Code">
69  private void initComponents() {
70
71      jScrollPane1 = new javax.swing.JScrollPane();
72      WordSet = new javax.swing.JList();
73      Edit = new javax.swing.JButton();
74      Add = new javax.swing.JButton();
75      Delete = new javax.swing.JButton();
76      Save = new javax.swing.JButton();
77      jLabel1 = new javax.swing.JLabel();
78      Title = new javax.swing.JTextField();
79      Cancel = new javax.swing.JButton();
80
81      setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
82
83      jScrollPane1.setViewportView(WordSet);
84
85      Edit.setText("Edit");
86      Edit.addActionListener(new java.awt.event.ActionListener() {
87          public void actionPerformed(java.awt.event.ActionEvent evt) {
88              EditActionPerformed(evt);
```

```
89          }
90      });
91
92      Add.setText("Add");
93      Add.addActionListener(new java.awt.event.ActionListener() {
94          public void actionPerformed(java.awt.event.ActionEvent evt) {
95              AddActionPerformed(evt);
96          }
97      });
98
99      Delete.setText("Delete");
100     Delete.addActionListener(new java.awt.event.ActionListener() {
101         public void actionPerformed(java.awt.event.ActionEvent evt) {
102             DeleteActionPerformed(evt);
103         }
104     });
105
106     Save.setText("Save");
107     Save.addActionListener(new java.awt.event.ActionListener() {
108         public void actionPerformed(java.awt.event.ActionEvent evt) {
109             SaveActionPerformed(evt);
110         }
111     });
112
113     jLabel1.setText("Title");
114
115     Title.addActionListener(new java.awt.event.ActionListener() {
116         public void actionPerformed(java.awt.event.ActionEvent evt) {
117             TitleActionPerformed(evt);
118         }
```

```
119     });
120
121     Cancel.setText("Cancel");
122     Cancel.addActionListener(new java.awt.event.ActionListener() {
123         public void actionPerformed(java.awt.event.ActionEvent evt) {
124             CancelActionPerformed(evt);
125         }
126     });
127
128     org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(getContentPane());
129     getContentPane().setLayout(layout);
130     layout.setHorizontalGroup(
131         layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
132         .add(layout.createSequentialGroup()
133             .addContainerGap()
134             .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
135                 .add(layout.createSequentialGroup()
136                     .add(0, 0, Short.MAX_VALUE)
137                     .add(jScrollPane1, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 329,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
138                     .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
139                     .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING, false)
140                         .add(Delete, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
141                         .add(Add, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 86,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
142                         .add(Save, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
143                         .add(Cancel)
```

```
144                     .add(Edit, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 86,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
145                 .add(layout.createSequentialGroup()
146                     .add(jLabel1)
147                     .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
148                     .add(Title)))
149             .addContainerGap())
150       );
151
152       layout.linkSize(new java.awt.Component[] {Add, Cancel, Delete, Edit, Save},
    org.jdesktop.layout.GroupLayout.HORIZONTAL);
153
154       layout.setVerticalGroup(
155           layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
156           .add(layout.createSequentialGroup()
157               .addContainerGap()
158               .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
159                   .add(layout.createSequentialGroup()
160                       .add(Add)
161                       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
162                       .add(Edit)
163                       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
164                       .add(Delete)
165                       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
166                       .add(Save)
167                       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
168                       .add(Cancel))
169                   .add(jScrollPane1))
170               .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
```

```
171             .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
172                 .add(Title, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
173                 .add(jLabel1))
174             .addContainerGap())
175     );
176 
177     layout.linkSize(new java.awt.Component[] {Add, Cancel, Delete, Edit, Save},
    org.jdesktop.layout.GroupLayout.VERTICAL);
178 
179     pack();
180 }// </editor-fold>
181 
182 private void TitleActionPerformed(java.awt.event.ActionEvent evt) {
183     // TODO add your handling code here:
184 }
185 
186 //allows the user to add words to the set by directing to the addWordform
187 private void AddActionPerformed(java.awt.event.ActionEvent evt) {
188     //opens the addword window
189     AddWordForm aw = new AddWordForm(this, true, phoneticLanguage);
190     aw.setVisible(true);
191     //increments the number of words and stores the word in the array
192     //Mastery Factor: Arrays
193     wordArray[numWords++] = aw.getWord();
194     //updates the list field
195     //Mastery Factor: Additional Java Libraries
196     WordSet.setListData(Arrays.copyOfRange(wordArray, 0, numWords));
197 }
198 
```

```
199   //allows the user to edit words in the set
200   private void EditActionPerformed(java.awt.event.ActionEvent evt) {
201       //prevents editing without selecting the word
202       if (WordSet.getSelectedIndex() == -1) {
203           JOptionPane.showMessageDialog(null, "You must select a word to edit");
204       } else {
205           //opens the edit word form
206           AddWordForm aw;
207           Word selectedWord = wordArray[WordSet.getSelectedIndex()];
208           if (phoneticLanguage) {
209               aw = new AddWordForm(this, true, (WordWPron) selectedWord, phoneticLanguage);
210           } else {
211               aw = new AddWordForm(this, true, (WordWOPron) selectedWord, phoneticLanguage);
212           }
213           aw.setVisible(true);
214           //updates
215           wordArray[WordSet.getSelectedIndex()] = aw.getWord();
216           WordSet.setListData(Arrays.copyOfRange(wordArray, 0, numWords));
217       }
218   }
219
220   //saves the vocabulary set
221   private void SaveActionPerformed(java.awt.event.ActionEvent evt) {
222       //stores the title of the vocabulary set
223       title = Title.getText();
224       if (title.equals("")) {
225           //prevents saving without a title
226           JOptionPane.showMessageDialog(null, "Please indicate the title of the vocabulary set");
227       } else if (title.length() > AppLogic.TITLE_RECORD_SIZE) {
228           //limits the number of characters that user can for the title
```

```
229         JOptionPane.showMessageDialog(null, "Please limit the title of your vocabulary set to "
230                 + AppLogic.TITLE_RECORD_SIZE + " characters");
231     } else if (numWords == 0) {
232         //prevents saving if no words in the set
233         JOptionPane.showMessageDialog(null, "You have not entered any words to the set");
234     } else {
235         title = Title.getText();
236         //closes the form
237         this.dispose();
238     }
239 }
240 
241 //closes the form
242 private void CancelActionPerformed(java.awt.event.ActionEvent evt) {
243     this.dispose();
244 }
245 
246 //deletes the word in the set
247 private void DeleteActionPerformed(java.awt.event.ActionEvent evt) {
248     //prevents deletion without specifying which word
249     if (WordSet.getSelectedIndex() == -1) {
250         JOptionPane.showMessageDialog(null, "You must select a word to delete");
251     } else {
252         if (WordSet.getSelectedIndex() + 1 == numWords) {
253             //deletes the last member of the word array
254             wordArray[WordSet.getSelectedIndex()] = null;
255         } else {
256             //rearranges the array
257             for (int i = WordSet.getSelectedIndex(); i < numWords - 1; i++) {
258                 wordArray[i] = wordArray[i + 1];
```

```
259             }
260             //deletes the last member
261             wordArray[numWords - 1] = null;
262           }
263           //decrements number of words
264           numWords--;
265       }
266       //updates the list field
267       WordSet.setListData(Arrays.copyOfRange(wordArray, 0, numWords));
268   }
269   /**
270    * @param args the command line arguments
271    */
272   // Variables declaration - do not modify
273   private javax.swing.JButton Add;
274   private javax.swing.JButton Cancel;
275   private javax.swing.JButton Delete;
276   private javax.swing.JButton Edit;
277   private javax.swing.JButton Save;
278   private javax.swing.JTextField Title;
279   private javax.swing.JList WordSet;
280   private javax.swing.JLabel jLabel1;
281   private javax.swing.JScrollPane jScrollPane1;
282   // End of variables declaration
283 }
284
```

AddWordForm.java

```
 1
 2 import javax.swing.JOptionPane;
 3
 4 /*
 5  * To change this template, choose Tools | Templates
 6  * and open the template in the editor.
 7  */
 8 /**
 9  *
10  * @author louienicholaslee, June 2012, International School Manila 13-inch
11  * MacBook Pro, NetBeans, To help facilitate learning of new vocabulary in
12  * foreign languages
13  *
14  * Allows user to edit/add words to the set
15  */
16 public class AddWordForm extends javax.swing.JDialog {
17     //declaration WordWPron/WorWOPron to send back to the edistSetForm
18
19     private WordWPron wp;
20     private WordWOPron wop;
21     private boolean phoneticLanguage;
22
23     /**
24      * Creates new form AddWord constructor for just adding a word
25      */
26     public AddWordForm(java.awt.Dialog parent, boolean modal, boolean phoneticLanguage) {
27         super(parent, modal);
28         initComponents();
```

```
29      this.phoneticLanguage = phoneticLanguage;
30      if (!phoneticLanguage) {
31          //sets the pronunciation field uneditable if the language is not phonetic based
32          Pronunciation.setEditable(false);
33      }
34      //shows a reminder to the user that there is a limit to the image size
35      JOptionPane.showMessageDialog(null, "Reminder: the maximum image size you"
36              + " the program can accommodate is 500 by 500 pixels");
37  }
38
39  //Mastery Factor: Polymorphism
40  //constructor for editing a word which passes a parameter WordWPron
41  public AddWordForm(java.awt.Dialog parent, boolean modal, WordWPron word, boolean phoneticLanguage) {
42      super(parent, modal);
43      initComponents();
44      wp = word;
45      this.phoneticLanguage = phoneticLanguage;
46      //enteres the fields with the current properties of the word
47      Word.setText(word.getWord());
48      Meaning.setText(word.getMeaning());
49      Example.setText(word.getExample());
50      ImgName.setText(word.getImgName());
51      Pronunciation.setText(word.getPronunciation());
52      //shows a reminder to the user that there is a limit to the image size
53      JOptionPane.showMessageDialog(null, "Reminder: the maximum image size you"
54              + " the program can accommodate is 500 by 500 pixels");
55  }
56
57  public AddWordForm(java.awt.Dialog parent, boolean modal, WordWOPron word, boolean phoneticLanguage) {
58      super(parent, modal);
```

```
59      initComponents();
60      wop = word;
61      this.phoneticLanguage = phoneticLanguage;
62      //enteres the fields with the current properties of the word
63      Word.setText(word.getWord());
64      Meaning.setText(word.getMeaning());
65      Example.setText(word.getExample());
66      ImgName.setText(word.getImgName());
67      //sets the pronunciation uneditable if the because it is not applicable
68      Pronunciation.setEditable(false);
69      //shows a reminder to the user that there is a limit to the image size
70      JOptionPane.showMessageDialog(null, "Reminder: the maximum image size you"
71              + " the program can accommodate is 500 by 500 pixels");
72
73  }
74
75  //accessor method that returns the proper word subclass based on whether or
76  //not the language is phonetic
77  public Word getWord() {
78      if (phoneticLanguage) {
79          return wp;
80      } else {
81          return wop;
82      }
83  }
84
85  /**
86   * This method is called from within the constructor to initialize the form.
87   * WARNING: Do NOT modify this code. The content of this method is always
88   * regenerated by the Form Editor.
```

```
89     */
90    @SuppressWarnings("unchecked")
91    // <editor-fold defaultstate="collapsed" desc="Generated Code">
92    private void initComponents() {
93
94        jDialog1 = new javax.swing.JDialog();
95        jLabel4 = new javax.swing.JLabel();
96        jTextField4 = new javax.swing.JTextField();
97        jLabel5 = new javax.swing.JLabel();
98        jTextField5 = new javax.swing.JTextField();
99        jLabel6 = new javax.swing.JLabel();
100       jTextField6 = new javax.swing.JTextField();
101       jLabel1 = new javax.swing.JLabel();
102       Word = new javax.swing.JTextField();
103       jLabel2 = new javax.swing.JLabel();
104       Meaning = new javax.swing.JTextField();
105       jLabel3 = new javax.swing.JLabel();
106       Pronunciation = new javax.swing.JTextField();
107       jLabel7 = new javax.swing.JLabel();
108       jLabel8 = new javax.swing.JLabel();
109       Example = new javax.swing.JTextField();
110       ImgName = new javax.swing.JTextField();
111       Enter = new javax.swing.JButton();
112       Cancel = new javax.swing.JButton();
113
114       jDialog1.setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
115
116       jLabel4.setText("English Word");
117
118       jTextField4.setText("jTextField1");
```

```
119
120     jLabel5.setText("Translated Word");
121
122     jTextField5.setText("jTextField2");
123
124     jLabel6.setText("Pronunciation");
125
126     jTextField6.setText("jTextField3");
127
128     org.jdesktop.layout.GroupLayout jDialog1Layout = new org.jdesktop.layout.GroupLayout(jDialog1.getContentPane());
129     jDialog1.getContentPane().setLayout(jDialog1Layout);
130     jDialog1Layout.setHorizontalGroup(
131        jDialog1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
132        .add(jDialog1Layout.createSequentialGroup()
133           .addContainerGap()
134           .add(jDialog1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
135              .add(jDialog1Layout.createSequentialGroup()
136                 .add(jLabel4)
137                 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
138                 .add(jTextField4))
139              .add(jDialog1Layout.createSequentialGroup()
140                 .add(jDialog1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
141                    .add(jDialog1Layout.createSequentialGroup()
142                       .add(jLabel5)
143                       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
144                       .add(jTextField5, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
145                    .add(jDialog1Layout.createSequentialGroup()
146                       .add(jLabel6)
147                       .add(40, 40, 40)
```

```
148                     .add(jTextField6, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
149                 .add(0, 177, Short.MAX_VALUE)))
150         .addContainerGap())
151     );
152     jDialog1Layout.setVerticalGroup(
153         jDialog1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
154         .add(jDialog1Layout.createSequentialGroup()
155             .addContainerGap()
156             .add(jDialog1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
157                 .add(jLabel4)
158                 .add(jTextField4, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
159             .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
160             .add(jDialog1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
161                 .add(jLabel5, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 16,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
162                 .add(jTextField5, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
163             .addPreferredGap(org.jdesktop.layout.LayoutStyle.UNRELATED)
164             .add(jDialog1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
165                 .add(jLabel6)
166                 .add(jTextField6, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
167             .addContainerGap(192, Short.MAX_VALUE))
168     );
169
170     setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
171     setResizable(false);
172
```

```
173     jLabel1.setText("Word");
174
175     Word.addActionListener(new java.awt.event.ActionListener() {
176         public void actionPerformed(java.awt.event.ActionEvent evt) {
177             WordActionPerformed(evt);
178         }
179     });
180
181     jLabel2.setText("Meaning");
182
183     jLabel3.setText("Pronunciation");
184
185     jLabel7.setText("Example");
186
187     jLabel8.setText("Image Name");
188
189     Enter.setText("Enter");
190     Enter.addActionListener(new java.awt.event.ActionListener() {
191         public void actionPerformed(java.awt.event.ActionEvent evt) {
192             EnterActionPerformed(evt);
193         }
194     });
195
196     Cancel.setText("Cancel");
197     Cancel.addActionListener(new java.awt.event.ActionListener() {
198         public void actionPerformed(java.awt.event.ActionEvent evt) {
199             CancelActionPerformed(evt);
200         }
201     });
202
```

```
203     org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(getContentPane());
204     getContentPane().setLayout(layout);
205     layout.setHorizontalGroup(
206       layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
207       .add(layout.createSequentialGroup()
208         .addContainerGap()
209         .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
210           .add(jLabel2)
211           .add(jLabel1)
212           .add(jLabel7)
213           .add(jLabel8))
214         .add(27, 27, 27)
215         .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
216           .add(Word, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 268,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
217           .add(Meaning, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 268,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
218           .add(Example, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
219           .add(ImgName, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
220         .add(0, 0, Short.MAX_VALUE))
221       .add(layout.createSequentialGroup()
222         .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
223           .add(layout.createSequentialGroup()
224             .addContainerGap()
225             .add(jLabel3)
226             .add(18, 18, 18)
227             .add(Pronunciation, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 268,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
```

```
228             .add(layout.createSequentialGroup()
229                 .add(133, 133, 133)
230                 .add(Enter)
231                 .add(39, 39, 39)
232                 .add(Cancel)))
233         .addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
234   );
235
236   layout.linkSize(new java.awt.Component[] {Example, ImgName, Meaning, Pronunciation, Word},
    org.jdesktop.layout.GroupLayout.HORIZONTAL);
237
238   layout.linkSize(new java.awt.Component[] {Cancel, Enter}, org.jdesktop.layout.GroupLayout.HORIZONTAL);
239
240   layout.setVerticalGroup(
241     layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
242     .add(layout.createSequentialGroup()
243       .addContainerGap()
244       .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
245         .add(jLabel1)
246         .add(Word, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
247       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
248       .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
249         .add(jLabel2, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 16,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
250         .add(Meaning, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
251       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
252       .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
253         .add(jLabel7)
```

```
254             .add(Example, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
255         .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
256         .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
257             .add(jLabel8)
258             .add(ImgName, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
259         .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
260         .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
261             .add(Pronunciation, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
262             .add(jLabel3))
263         .add(1, 1, 1)
264         .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
265             .add(Enter)
266             .add(Cancel))
267         .add(0, 0, Short.MAX_VALUE))
268     );
269
270     pack();
271 }// </editor-fold>
272
273 private void EnterActionPerformed(java.awt.event.ActionEvent evt) {
274     int imgNameLength = ImgName.getText().trim().length();
275     //prevention of the different possibilities of error by showing messages to user
276     //Mastery Factor: Complex Selection
277     if ("".equals(Word.getText())
278         && "".equals(Meaning.getText())
279         && "".equals(Example.getText())
280         && "".equals(ImgName.getText())
```

```
281             && "".equals(Pronunciation.getText())) {
282         JOptionPane.showMessageDialog(null,
283                 "You have not filled in the proper fields: Word, Meaning,"
284                 + " Pronunciation (optional), Example, and Image Name");
285     } else if ("".equals(Word.getText()) && "".equals(Meaning.getText())
286             && "".equals(Example.getText())
287             && "".equals(ImgName.getText())) {
288         JOptionPane.showMessageDialog(null,
289                 "You have not filled in the proper fields: Word, Meaning, Example, and Image Name");
290     } else if ("".equals(Word.getText())
291             && "".equals(Meaning.getText())
292             && "".equals(Example.getText())
293             && "".equals(Pronunciation.getText())) {
294         JOptionPane.showMessageDialog(null,
295                 "You have not filled in the proper fields: Word, Meaning,"
296                 + " Pronunciation (optional), and Example");
297     } else if ("".equals(Word.getText())
298             && "".equals(Meaning.getText())
299             && "".equals(Example.getText())) {
300         JOptionPane.showMessageDialog(null,
301                 "You have not filled in the proper fields: Word, Meaning, and Example");
302     } else if ("".equals(Word.getText())
303             && "".equals(Meaning.getText())
304             && "".equals(ImgName.getText())
305             && "".equals(Pronunciation.getText())) {
306         JOptionPane.showMessageDialog(null,
307                 "You have not filled in the proper fields: Word, Meaning,"
308                 + " Pronunciation (optional) and Image Name");
309     } else if ("".equals(Word.getText())
310             && "".equals(Meaning.getText())
```

```
311             && "".equals(ImgName.getText())) {
312         JOptionPane.showMessageDialog(null,
313                 "You have not filled in the proper fields: Word, Meaning, and Image Name");
314     } else if ("".equals(Word.getText())
315             && "".equals(Example.getText())
316             && "".equals(ImgName.getText())
317             && "".equals(Pronunciation.getText())) {
318         JOptionPane.showMessageDialog(null,
319                 "You have not filled in the proper fields: Word, Pronunciation (optional),"
320                 + " Example, and Image Name");
321     } else if ("".equals(Word.getText())
322             && "".equals(Example.getText())
323             && "".equals(ImgName.getText())) {
324         JOptionPane.showMessageDialog(null,
325                 "You have not filled in the proper fields: Word, Example, and Image Name");
326     } else if ("".equals(Meaning.getText())
327             && "".equals(Example.getText())
328             && "".equals(ImgName.getText())
329             && "".equals(Pronunciation.getText())) {
330         JOptionPane.showMessageDialog(null,
331                 "You have not filled in the proper fields: Meaning, Pronunciation (optional),"
332                 + " Example, and Image Name");
333     } else if ("".equals(Meaning.getText())
334             && "".equals(Example.getText())
335             && "".equals(ImgName.getText())) {
336         JOptionPane.showMessageDialog(null,
337                 "You have not filled in the proper fields: Meaning, Example, and Image Name");
338     } else if ("".equals(Word.getText())
339             && "".equals(Meaning.getText())
340             && "".equals(Pronunciation.getText())) {
```

```
341         JOptionPane.showMessageDialog(null,
342                 "You have not filled in the proper fields: Word, Meaning, and Pronunciation (optional)");
343     } else if ("".equals(Word.getText())
344             && "".equals(Meaning.getText())) {
345         JOptionPane.showMessageDialog(null,
346                 "You have not filled in the proper fields: Word and Meaning");
347     } else if ("".equals(Word.getText())
348             && "".equals(Example.getText())
349             && "".equals(Pronunciation.getText())) {
350         JOptionPane.showMessageDialog(null,
351                 "You have not filled in the proper fields: Word, Pronunciation (optional), and Example");
352     } else if ("".equals(Word.getText())
353             && "".equals(Example.getText())) {
354         JOptionPane.showMessageDialog(null,
355                 "You have not filled in the proper fields: Word and Example");
356     } else if ("".equals(Meaning.getText())
357             && "".equals(Example.getText())
358             && "".equals(Pronunciation.getText())) {
359         JOptionPane.showMessageDialog(null,
360                 "You have not filled in the proper fields: Meaning, Pronunciation (optional), and Example");
361     } else if ("".equals(Meaning.getText())
362             && "".equals(Example.getText())) {
363         JOptionPane.showMessageDialog(null,
364                 "You have not filled in the proper fields: Meaning and Example");
365     } else if ("".equals(Word.getText())
366             && "".equals(Example.getText())
367             && "".equals(Pronunciation.getText())) {
368         JOptionPane.showMessageDialog(null,
369                 "You have not filled in the proper fields: Word, Pronunciation (optional), and Example");
370     } else if ("".equals(Word.getText())
```

```
371             && "".equals(Example.getText())) {
372         JOptionPane.showMessageDialog(null,
373                 "You have not filled in the proper fields: Word and Example");
374     } else if ("".equals(Word.getText())
375             && "".equals(Meaning.getText())
376             && "".equals(Pronunciation.getText())) {
377         JOptionPane.showMessageDialog(null,
378                 "You have not filled in the proper fields: Word, Meaning, and Pronunciation (optional)");
379     } else if ("".equals(Word.getText())
380             && "".equals(Meaning.getText())) {
381         JOptionPane.showMessageDialog(null,
382                 "You have not filled in the proper fields: Word and Meaning");
383     } else if ("".equals(Word.getText())
384             && "".equals(ImgName.getText())
385             && "".equals(Pronunciation.getText())) {
386         JOptionPane.showMessageDialog(null,
387                 "You have not filled in the proper fields: Word, Pronunciation (optional), and Image Name");
388     } else if ("".equals(Word.getText())
389             && "".equals(ImgName.getText())) {
390         JOptionPane.showMessageDialog(null,
391                 "You have not filled in the proper fields: Word and Image Name");
392     } else if ("".equals(Meaning.getText())
393             && "".equals(ImgName.getText())
394             && "".equals(Pronunciation.getText())) {
395         JOptionPane.showMessageDialog(null,
396                 "You have not filled in the proper fields: Meaning, Pronunciation (optional), and Image Name");
397     } else if ("".equals(Meaning.getText())
398             && "".equals(ImgName.getText())) {
399         JOptionPane.showMessageDialog(null,
400                 "You have not filled in the proper fields: Meaning and Image Name");
```

```
401     } else if ("".equals(Meaning.getText())
402             && "".equals(ImgName.getText())
403             && "".equals(Pronunciation.getText())) {
404         JOptionPane.showMessageDialog(null,
405                 "You have not filled in the proper fields: Meaning, and Image Name");
406     } else if ("".equals(Example.getText())
407             && "".equals(ImgName.getText())) {
408         JOptionPane.showMessageDialog(null,
409                 "You have not filled in the proper fields: Example and Image Name");
410     } else if ("".equals(Example.getText())
411             && "".equals(ImgName.getText())
412             && "".equals(Pronunciation.getText())) {
413         JOptionPane.showMessageDialog(null,
414                 "You have not filled in the proper fields: Pronunciation (optional), Example, and Image Name");
415     } else if ("".equals(Word.getText())
416             && "".equals(Pronunciation.getText())) {
417         JOptionPane.showMessageDialog(null,
418                 "You have not filled in the proper fields: Word and Pronunciation (optional)");
419     } else if ("".equals(Word.getText())) {
420         JOptionPane.showMessageDialog(null,
421                 "Please enter the word");
422     } else if ("".equals(Meaning.getText())
423             && "".equals(Pronunciation.getText())) {
424         JOptionPane.showMessageDialog(null,
425                 "You have not filled in the proper fields: Meaning and Pronunciation (optional)");
426     } else if ("".equals(Meaning.getText())) {
427         JOptionPane.showMessageDialog(null,
428                 "Please enter the meaning of the word");
429     } else if ("".equals(Example.getText())
430             && "".equals(Pronunciation.getText())) {
```

```
431         JOptionPane.showMessageDialog(null,
432                 "You have not filled in the proper fields: Pronunciation (optional) and Example");
433       } else if ("".equals(Example.getText())) {
434         JOptionPane.showMessageDialog(null, "Please enter an example usage of the word");
435         //prevents user from entering an image file name that is not currently
436         //in the imageDirectory
437       } else if (this.getClass().getResource("/ImageDirectory/" + ImgName.getText()) == null) {
438         JOptionPane.showMessageDialog(null, "Please enter a valid image file name. "
439                 + "Note: you must first enter image file to the ImageDirectory folder");
440         //forces user to print image of type jpeg, jpg or gif
441       } else if (!ImgName.getText().trim().substring(imgNameLength- 3, imgNameLength).equalsIgnoreCase("jpg")
442             && !ImgName.getText().trim().substring(imgNameLength- 3, imgNameLength).equalsIgnoreCase("gif")
443             && !ImgName.getText().trim().substring(imgNameLength- 4, imgNameLength).equalsIgnoreCase("jpeg")) {
444         JOptionPane.showMessageDialog(null, "Please enter a valid image file name."
445                 + " Note: the picture must be of file type .jpg, .jpeg, or .gif");
446       } else {
447         //creates instantiates the correct word subclass
448         if (phoneticLanguage) {
449           //prevents user from only entering spaces in the fields
450           if (Word.getText().trim().length() == 0
451                 || Meaning.getText().trim().length() == 0
452                 || Pronunciation.getText().trim().length() == 0
453                 || Example.getText().trim().length() == 0
454                 || ImgName.getText().trim().length() == 0) {
455             JOptionPane.showMessageDialog(null, "At least one of the fields contain only spaces");
456           } else {
457             wp = new WordWPron(Word.getText().trim(), Meaning.getText().trim(),
458                   Example.getText().trim(), ImgName.getText().trim(), Pronunciation.getText().trim());
459             this.dispose();
460           }
```

```
461         } else {
462             //prevents user from only entering spaces in the fields
463             if (Word.getText().trim().length() == 0
464                     || Meaning.getText().trim().length() == 0
465                     || Example.getText().trim().length() == 0
466                     || ImgName.getText().trim().length() == 0) {
467                 JOptionPane.showMessageDialog(null, "At least one of the fields contain only spaces");
468             } else {
469                 wop = new WordWOPron(Word.getText().trim(), Meaning.getText().trim(),
470                         Example.getText().trim(), ImgName.getText().trim());
471                 this.dispose();
472             }
473         }
474     }
475 }
476 //closes the form
477 private void CancelActionPerformed(java.awt.event.ActionEvent evt) {
478     this.dispose();
479 }
480 
481 private void WordActionPerformed(java.awt.event.ActionEvent evt) {
482     // TODO add your handling code here:
483 }
484 /**
485  * @param args the command line arguments
486  */
487 // Variables declaration - do not modify
488 private javax.swing.JButton Cancel;
489 private javax.swing.JButton Enter;
490 private javax.swing.JTextField Example;
```

```
491     private javax.swing.JTextField ImgName;
492     private javax.swing.JTextField Meaning;
493     private javax.swing.JTextField Pronunciation;
494     private javax.swing.JTextField Word;
495     private javax.swing.JDialog jDialog1;
496     private javax.swing.JLabel jLabel1;
497     private javax.swing.JLabel jLabel2;
498     private javax.swing.JLabel jLabel3;
499     private javax.swing.JLabel jLabel4;
500     private javax.swing.JLabel jLabel5;
501     private javax.swing.JLabel jLabel6;
502     private javax.swing.JLabel jLabel7;
503     private javax.swing.JLabel jLabel8;
504     private javax.swing.JTextField jTextField4;
505     private javax.swing.JTextField jTextField5;
506     private javax.swing.JTextField jTextField6;
507     // End of variables declaration
508 }
509
```

TestForm.java

```
 1
 2 import javax.swing.ImageIcon;
 3 import javax.swing.JOptionPane;
 4
 5 /*
 6  * To change this template, choose Tools | Templates
 7  * and open the template in the editor.
 8  */
 9 /**
10  * @author louienicholaslee, June 2012, International School Manila 13-inch
11  * MacBook Pro, NetBeans. Purpose: To help facilitate learning of new vocabulary in
12  * foreign languages
13  *
14  * Tests the user on preexisting vocabulary set
15  */
16 public class TestForm extends javax.swing.JDialog {
17     //shows the current position in the word array, ie which word is being tested
18     private int currentIndex = 0;
19     //Stores the array of words in the vocabulary set being tested
20     private Word[] wordArray = null;
21     //stores the incorrectly answered words. will be used to populate the view
22     //form to show the solutions to incorrectly answered questions
23     private Word[] incorrectlyAnswered = null;
24     private boolean phoneticLanguage;
25     //Stores the answers of the user
26     private String[] answers = null;
27     //stores the number of questions that will be tested
28     private int numQuestions;
```

```
29  // stores the number of incorrect answers that user has
30  private int numWrong = 0;
31
32  /**
33   * Creates new form TestForm
34   */
35  public TestForm(java.awt.Frame parent, boolean modal, Word[] wordArray,
36          boolean phoneticLanguage, String title) {
37      super(parent, modal);
38      initComponents();
39      this.wordArray = wordArray;
40      //number of questions is equal to the number of words in the array
41      numQuestions = wordArray.length;
42      //the number of answers must correspond to the number of questions
43      answers = new String[numQuestions];
44      //sets the answers as blanks instead of null to avoid nullpointerexcpetion error
45      for (int i = 0; i < numQuestions; i++) {
46          answers[i] = "";
47      }
48      this.phoneticLanguage = phoneticLanguage;
49      //indicates the vocabulary set being tested
50      VocabSetTitle.setText(title);
51      //sets the word field as editable so the the user can input their answer
52      Word.setEditable(true);
53      //sets other fields as uneditable
54      Meaning.setEditable(false);
55      Example.setEditable(false);
56      Pronunciation.setEditable(false);
57      //sets the Word field blank
58      Word.setText(null);
```

```
59      //enters the correct information about the word in the form
60      Meaning.setText(wordArray[currentIndex].getMeaning());
61      Example.setText(wordArray[currentIndex].getExample());
62      //shows the image of the word
63      Image.setIcon(new ImageIcon(this.getClass().getResource("/ImageDirectory/"+
64            wordArray[currentIndex].getImgName())));
65      //adds the pronunciation if applicable
66      if (phoneticLanguage) {
67         Pronunciation.setText(((WordWPron) wordArray[currentIndex]).getPronunciation());
68      } else {
69         Pronunciation.setText("Not Available");
70      }
71   }
72
73 //-------------------------------Accessor Methods-----------------------------//
74   public Word[] getIncorrectAnswers() {
75      return incorrectlyAnswered;
76   }
77
78   public int getNumWrong() {
79      return numWrong;
80   }
81
82   public int getNumQuestions(){
83      return numQuestions;
84   }
85 //-----------------------------------------------------------------------------//
86
87   /**
88    * This method is called from within the constructor to initialize the form.
```

```
89     * WARNING: Do NOT modify this code. The content of this method is always
90     * regenerated by the Form Editor.
91     */
92    @SuppressWarnings("unchecked")
93    // <editor-fold defaultstate="collapsed" desc="Generated Code">
94    private void initComponents() {
95
96        Done = new javax.swing.JButton();
97        VocabSetTitle = new javax.swing.JLabel();
98        Next = new javax.swing.JButton();
99        Previous = new javax.swing.JButton();
100       jLabel2 = new javax.swing.JLabel();
101       Example = new javax.swing.JTextField();
102       jSeparator2 = new javax.swing.JSeparator();
103       jLabel3 = new javax.swing.JLabel();
104       Image = new javax.swing.JLabel();
105       jLabel5 = new javax.swing.JLabel();
106       Meaning = new javax.swing.JTextField();
107       Word = new javax.swing.JTextField();
108       jLabel6 = new javax.swing.JLabel();
109       Pronunciation = new javax.swing.JTextField();
110       jLabel4 = new javax.swing.JLabel();
111
112       setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
113
114       Done.setText("Done");
115       Done.addActionListener(new java.awt.event.ActionListener() {
116           public void actionPerformed(java.awt.event.ActionEvent evt) {
117               DoneActionPerformed(evt);
118           }
```

```
119     });
120
121     VocabSetTitle.setFont(new java.awt.Font("Lucida Grande", 0, 24)); // NOI18N
122     VocabSetTitle.setText("Nouns");
123     VocabSetTitle.setVerticalAlignment(javax.swing.SwingConstants.TOP);
124
125     Next.setText("Next");
126     Next.addActionListener(new java.awt.event.ActionListener() {
127         public void actionPerformed(java.awt.event.ActionEvent evt) {
128             NextActionPerformed(evt);
129         }
130     });
131
132     Previous.setText("Previous");
133     Previous.addActionListener(new java.awt.event.ActionListener() {
134         public void actionPerformed(java.awt.event.ActionEvent evt) {
135             PreviousActionPerformed(evt);
136         }
137     });
138
139     jLabel2.setText("Word");
140
141     Example.setEditable(false);
142     Example.setText("我很快樂");
143
144     jSeparator2.setOrientation(javax.swing.SwingConstants.VERTICAL);
145
146     jLabel3.setText("Meaning");
147
148     Image.setMaximumSize(new java.awt.Dimension(100, 100));
```

```
149     Image.setMinimumSize(new java.awt.Dimension(100, 100));
150     Image.setPreferredSize(new java.awt.Dimension(100, 100));
151
152     jLabel5.setFont(new java.awt.Font("Hobo Std", 0, 70)); // NOI18N
153     jLabel5.setText("Test ");
154
155     Meaning.setEditable(false);
156     Meaning.setText("快樂");
157
158     Word.setEditable(false);
159     Word.setText("Happy");
160
161     jLabel6.setText("Pronunciation");
162
163     Pronunciation.setEditable(false);
164     Pronunciation.setText("Kuai Le");
165
166     jLabel4.setText("Example");
167
168     org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(getContentPane());
169     getContentPane().setLayout(layout);
170     layout.setHorizontalGroup(
171         layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
172         .add(layout.createSequentialGroup()
173             .addContainerGap()
174             .add(Image, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 500,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
175             .addPreferredGap(org.jdesktop.layout.LayoutStyle.UNRELATED)
176             .add(jSeparator2, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 12,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
```

```
177         .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
178           .add(layout.createSequentialGroup()
179             .add(6, 6, 6)
180             .add(Previous)
181             .add(149, 149, 149)
182             .add(Next, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 97,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
183             .add(146, 146, 146)
184             .add(Done, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 97,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
185             .add(76, 76, 76))
186           .add(layout.createSequentialGroup()
187             .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
188             .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
189               .add(layout.createSequentialGroup()
190                 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
191                   .add(jLabel3)
192                   .add(jLabel2)
193                   .add(jLabel6)
194                   .add(jLabel4))
195                 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
196                 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
197                   .add(org.jdesktop.layout.GroupLayout.TRAILING, Meaning,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 353, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
198                   .add(org.jdesktop.layout.GroupLayout.TRAILING, Word)
199                   .add(org.jdesktop.layout.GroupLayout.TRAILING, Pronunciation,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 353, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
200                   .add(org.jdesktop.layout.GroupLayout.TRAILING, Example,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 563, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
201               .add(layout.createSequentialGroup()
```

```
202                     .add(jLabel5)
203                     .add(0, 0, Short.MAX_VALUE)))
204                 .addContainerGap())
205             .add(org.jdesktop.layout.GroupLayout.TRAILING, layout.createSequentialGroup()
206                 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
207                 .add(VocabSetTitle, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
208                 .addContainerGap())))
209     );
210 
211     layout.linkSize(new java.awt.Component[] {Example, Meaning, Pronunciation},
    org.jdesktop.layout.GroupLayout.HORIZONTAL);
212 
213     layout.setVerticalGroup(
214         layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
215         .add(layout.createSequentialGroup()
216             .addContainerGap()
217             .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
218                 .add(layout.createSequentialGroup()
219                     .add(jLabel5)
220                     .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
221                     .add(VocabSetTitle, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
222                     .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
223                     .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
224                         .add(jLabel2)
225                         .add(Word, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
226                     .add(31, 31, 31)
227                     .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
```

```
228                 .add(Meaning, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
229                 .add(jLabel3))
230               .add(26, 26, 26)
231               .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
232                 .add(Pronunciation, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
233                 .add(jLabel6))
234               .add(23, 23, 23)
235               .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
236                 .add(Example, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
237                 .add(jLabel4, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 16,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
238               .add(20, 20, 20)
239               .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
240                 .add(Previous)
241                 .add(Next)
242                 .add(Done))
243               .add(36, 36, 36))
244             .add(layout.createSequentialGroup()
245               .add(Image, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 500,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
246               .add(0, 0, Short.MAX_VALUE))
247             .add(layout.createSequentialGroup()
248               .add(jSeparator2)
249               .addContainerGap())))
250     );
251
252     pack();
```

```
253   }// </editor-fold>
254
255   //the user presses this button to indicate that they are finished with the
256   //test. The number of incorrects is counted and the incorrectlyAnswered word
257   //array is populated. The test form is then closed
258   private void DoneActionPerformed(java.awt.event.ActionEvent evt) {
259      //A temporary word array that will be used to store the words that have
260      //not yet been memorized, ie ones that have the wrong answers
261      Word[] tmp = new Word[numQuestions];
262      //ensures that the answer to the currect question is saved
263      answers[currentIndex] = Word.getText().trim();
264      //populates the tmp word array and calcates the number of mistakes
265      for (int i = 0; i < numQuestions; i++) {
266         if (!answers[i].trim().equalsIgnoreCase(wordArray[i].getWord())) {
267            tmp[numWrong++] = wordArray[i];
268            //adds the word not yet memorized if the question corresponding
269            //to the word is not correctly answered and if the word is not
270            //already in the vocabulary set
271            //protects from duplicating
272            if (!AppLogic.wordsNotMemorized.find(wordArray[i])) {
273               AppLogic.wordsNotMemorized.insert(wordArray[i]);
274            }
275         } else {
276            //removes the words from the words not yet memorized set if the
277            //user answers that question for the word correctly and if the
278            //word was in the set to begin with
279            if (AppLogic.wordsNotMemorized.find(wordArray[i])) {
280               AppLogic.wordsNotMemorized.remove(wordArray[i]);
281            }
282         }
```

```
283     }
284     //stores the words with questions answered incorrectly to the incorrectlyAnswered
285     //word array
286
287     if (numWrong > 0) {
288         incorrectlyAnswered = new Word[numWrong];
289         for (int j = 0; j < numWrong; j++) {
290             incorrectlyAnswered[j] = tmp[j];
291         }
292     }
293     //writes the new wordTree to file
294     AppLogic.deleteSet("Words Not Yet Memorized.txt");
295     AppLogic.wordsNotMemorized.saveWordTree();
296     //closes the form
297     this.dispose();
298 }
299
300 //shows the next word in the test and stores the answer
301 private void NextActionPerformed(java.awt.event.ActionEvent evt) {
302     //shows a message that tells the reader that they can't go back any more
303     //due to there being no more words
304     if (currentIndex < wordArray.length - 1) {
305         //stores the answers
306         answers[currentIndex] = Word.getText().trim();
307         //increments the current index to indicate the position
308         currentIndex++;
309         //rewrites the form to include the information on the next word
310         Word.setText(answers[currentIndex]);
311         Meaning.setText(wordArray[currentIndex].getMeaning());
312         Example.setText(wordArray[currentIndex].getExample());
```

```
313         Image.setIcon(new ImageIcon(this.getClass().getResource("/ImageDirectory/" +
314                 wordArray[currentIndex].getImgName())));
315         if (phoneticLanguage) {
316             Pronunciation.setText(((WordWPron) wordArray[currentIndex]).getPronunciation());
317         } else {
318             Pronunciation.setText("Not Available");
319         }
320     } else {
321         JOptionPane.showMessageDialog(null, "You have reached the end of the test. "
322                 + "You can go back to previous questions by pressing the previous button "
323                 + "or finish the test by pressing the done button.");
324     }
325 }
326 
327 //shows the previous word in the test and stores the answer
328 private void PreviousActionPerformed(java.awt.event.ActionEvent evt) {
329     //shows a message that tells the reader that they can't go back any more
330     //due to there being no more words
331     if (currentIndex > 0) {
332         //stores the answer
333         answers[currentIndex] = Word.getText().trim();
334         //decrements the current indext to indicate the position
335         currentIndex--;
336         //rewrites the form to include the information on the next word
337         Word.setText(answers[currentIndex]);
338         Meaning.setText(wordArray[currentIndex].getMeaning());
339         Example.setText(wordArray[currentIndex].getExample());
340         Image.setIcon(new ImageIcon(this.getClass().getResource("/ImageDirectory/" +
341                 wordArray[currentIndex].getImgName())));
342         if (phoneticLanguage) {
```

```
343             Pronunciation.setText(((WordWPron) wordArray[currentIndex]).getPronunciation());
344         } else {
345             Pronunciation.setText("Not Available");
346         }
347     } else {
348         JOptionPane.showMessageDialog(null, "There are no more words that come before this word");
349     }
350 }
351 /**
352  * @param args the command line arguments
353  */
354 // Variables declaration - do not modify
355 private javax.swing.JButton Done;
356 private javax.swing.JTextField Example;
357 private javax.swing.JLabel Image;
358 private javax.swing.JTextField Meaning;
359 private javax.swing.JButton Next;
360 private javax.swing.JButton Previous;
361 private javax.swing.JTextField Pronunciation;
362 private javax.swing.JLabel VocabSetTitle;
363 private javax.swing.JTextField Word;
364 private javax.swing.JLabel jLabel2;
365 private javax.swing.JLabel jLabel3;
366 private javax.swing.JLabel jLabel4;
367 private javax.swing.JLabel jLabel5;
368 private javax.swing.JLabel jLabel6;
369 private javax.swing.JSeparator jSeparator2;
370 // End of variables declaration
371 }
372
```

ViewForm.java

```
 1
 2 import javax.swing.ImageIcon;
 3 import javax.swing.JOptionPane;
 4
 5 /*
 6  * To change this template, choose Tools | Templates
 7  * and open the template in the editor.
 8  */
 9 /**
10  * @author louienicholaslee, June 2012, International School Manila 13-inch
11  * MacBook Pro, NetBeans. Purpose: To help facilitate learning of new vocabulary in
12  * foreign languages
13  *
14  * Views the preexisting vocabulary one word at a time
15  */
16 public class ViewForm extends javax.swing.JDialog {
17     //shows the current position in the word array, ie which word is being viewed
18     private int currentIndex = 0;
19     //stores the array of words that will be viewed
20     private Word[] wordArray = null;
21     private boolean phoneticLanguage;
22
23     /**
24      * Creates new form ViewForm Constructor that is used to view specific
25      * vocabulary sets that that user has created
26      */
27     public ViewForm(java.awt.Frame parent, boolean modal,
28             Word[] wordArray, boolean phoneticLanguage, String title) {
```

```
29      super(parent, modal);
30      initComponents();
31      this.wordArray = wordArray;
32      this.phoneticLanguage = phoneticLanguage;
33      //shows the title of the vocabulary set
34      VocabSetTitle.setText(title);
35      //makes the text fields uneditable
36      Word.setEditable(false);
37      Meaning.setEditable(false);
38      Example.setEditable(false);
39      Pronunciation.setEditable(false);
40      //enters the correct information on the word to the form
41      Word.setText(wordArray[0].getWord());
42      Meaning.setText(wordArray[0].getMeaning());
43      Example.setText(wordArray[0].getExample());
44      //shows the picture
45      Image.setIcon(new ImageIcon(this.getClass().getResource("/ImageDirectory/" +
46              wordArray[0].getImgName())));
47      //displays the pronunciation if applicable
48      if (phoneticLanguage) {
49         Pronunciation.setText(((WordWPron) wordArray[0]).getPronunciation());
50      } else {
51         Pronunciation.setText("Not Available");
52      }
53  }
54
55  //Constructor that is used to show the solutions to the incorrectly answered
56  //questions
57  public ViewForm(java.awt.Frame parent, boolean modal, Word[] wordArray, boolean phoneticLanguage) {
58      super(parent, modal);
```

```
59      initComponents();
60      this.wordArray = wordArray;
61      this.phoneticLanguage = phoneticLanguage;
62      //corrects the labels
63      Label.setText("Solutions");
64      VocabSetTitle.setText("Incorrect Answers");
65      //makes the text fields uneditable
66      Word.setEditable(false);
67      Meaning.setEditable(false);
68      Example.setEditable(false);
69      Pronunciation.setEditable(false);
70      //enters the correct information on the word to the form
71      Word.setText(wordArray[0].getWord());
72      Meaning.setText(wordArray[0].getMeaning());
73      Example.setText(wordArray[0].getExample());
74      //shows the picture
75      Image.setIcon(new ImageIcon(this.getClass().getResource("/ImageDirectory/" +
76              wordArray[0].getImgName())));
77      //displays the pronunciation if applicable
78      if (phoneticLanguage) {
79         Pronunciation.setText(((WordWPron) wordArray[0]).getPronunciation());
80      } else {
81         Pronunciation.setText("Not Available");
82      }
83  }
84
85  /**
86   * This method is called from within the constructor to initialize the form.
87   * WARNING: Do NOT modify this code. The content of this method is always
88   * regenerated by the Form Editor.
```

```
89     */
90    @SuppressWarnings("unchecked")
91    // <editor-fold defaultstate="collapsed" desc="Generated Code">
92    private void initComponents() {
93
94        jLabel3 = new javax.swing.JLabel();
95        jSeparator2 = new javax.swing.JSeparator();
96        jLabel6 = new javax.swing.JLabel();
97        Word = new javax.swing.JTextField();
98        Meaning = new javax.swing.JTextField();
99        Label = new javax.swing.JLabel();
100       jLabel4 = new javax.swing.JLabel();
101       Pronunciation = new javax.swing.JTextField();
102       Image = new javax.swing.JLabel();
103       jLabel2 = new javax.swing.JLabel();
104       Example = new javax.swing.JTextField();
105       Next = new javax.swing.JButton();
106       Previous = new javax.swing.JButton();
107       Exit = new javax.swing.JButton();
108       VocabSetTitle = new javax.swing.JLabel();
109
110       setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
111
112       jLabel3.setText("Meaning");
113
114       jSeparator2.setOrientation(javax.swing.SwingConstants.VERTICAL);
115
116       jLabel6.setText("Pronunciation");
117
118       Word.setEditable(false);
```

```
119     Word.setText("Happy");
120
121     Meaning.setEditable(false);
122     Meaning.setText("快樂");
123
124     Label.setFont(new java.awt.Font("Hobo Std", 0, 70)); // NOI18N
125     Label.setText("View Words");
126
127     jLabel4.setText("Example");
128
129     Pronunciation.setEditable(false);
130     Pronunciation.setText("Kuai Le");
131
132     Image.setMaximumSize(new java.awt.Dimension(100, 100));
133     Image.setMinimumSize(new java.awt.Dimension(100, 100));
134     Image.setPreferredSize(new java.awt.Dimension(100, 100));
135
136     jLabel2.setText("Word");
137
138     Example.setEditable(false);
139     Example.setText("我很快樂");
140
141     Next.setText("Next");
142     Next.addActionListener(new java.awt.event.ActionListener() {
143         public void actionPerformed(java.awt.event.ActionEvent evt) {
144             NextActionPerformed(evt);
145         }
146     });
147
148     Previous.setText("Previous");
```

```
149     Previous.addActionListener(new java.awt.event.ActionListener() {
150         public void actionPerformed(java.awt.event.ActionEvent evt) {
151             PreviousActionPerformed(evt);
152         }
153     });
154 
155     Exit.setText("Exit");
156     Exit.addActionListener(new java.awt.event.ActionListener() {
157         public void actionPerformed(java.awt.event.ActionEvent evt) {
158             ExitActionPerformed(evt);
159         }
160     });
161 
162     VocabSetTitle.setFont(new java.awt.Font("Lucida Grande", 0, 24)); // NOI18N
163     VocabSetTitle.setText("Nouns");
164     VocabSetTitle.setVerticalAlignment(javax.swing.SwingConstants.TOP);
165 
166     org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(getContentPane());
167     getContentPane().setLayout(layout);
168     layout.setHorizontalGroup(
169         layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
170         .add(layout.createSequentialGroup()
171             .addContainerGap()
172             .add(Image, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 500,
        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
173             .addPreferredGap(org.jdesktop.layout.LayoutStyle.UNRELATED)
174             .add(jSeparator2, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 12,
        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
175             .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
176                 .add(layout.createSequentialGroup()
```

```
177                 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
178                 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
179                   .add(layout.createSequentialGroup()
180                     .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
181                       .add(jLabel3)
182                       .add(jLabel2)
183                       .add(jLabel6)
184                       .add(jLabel4))
185                     .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
186                     .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
187                       .add(org.jdesktop.layout.GroupLayout.TRAILING, Word,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
188                       .add(Example, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 536,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
189                       .add(org.jdesktop.layout.GroupLayout.TRAILING, Meaning,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 353, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
190                       .add(org.jdesktop.layout.GroupLayout.TRAILING, Pronunciation,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 353, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
191                   .add(Label))
192                 .addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
193               .add(layout.createSequentialGroup()
194                 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
195                 .add(VocabSetTitle, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
196                 .addContainerGap())
197               .add(layout.createSequentialGroup()
198                 .add(6, 6, 6)
199                 .add(Previous)
200                 .add(115, 115, 115)
```

```
201                 .add(Next, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 97,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
202                 .add(132, 132, 132)
203                 .add(Exit, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 97,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
204                 .add(103, 103, 103))))
205       );
206
207       layout.linkSize(new java.awt.Component[] {Example, Meaning, Pronunciation, Word},
    org.jdesktop.layout.GroupLayout.HORIZONTAL);
208
209       layout.setVerticalGroup(
210           layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
211           .add(layout.createSequentialGroup()
212               .addContainerGap()
213               .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
214                   .add(layout.createSequentialGroup()
215                       .add(Label)
216                       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
217                       .add(VocabSetTitle, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
218                       .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
219                       .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
220                           .add(jLabel2)
221                           .add(Word, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
222                       .add(31, 31, 31)
223                       .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
224                           .add(Meaning, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
```

```
225                     .add(jLabel3))
226                 .add(26, 26, 26)
227                 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
228                     .add(Pronunciation, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
229                     .add(jLabel6))
230                 .add(23, 23, 23)
231                 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
232                     .add(Example, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
    org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
233                     .add(jLabel4, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 16,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
234                 .add(20, 20, 20)
235                 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
236                     .add(Previous)
237                     .add(Next)
238                     .add(Exit))
239                 .add(36, 36, 36))
240             .add(layout.createSequentialGroup()
241                 .add(Image, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 500,
    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
242                 .add(0, 1, Short.MAX_VALUE))
243             .add(layout.createSequentialGroup()
244                 .add(jSeparator2)
245                 .addContainerGap())))
246     );
247
248     pack();
249 }// </editor-fold>
250
```

```
251   //shows the previous word in the vocabulary set
252   private void PreviousActionPerformed(java.awt.event.ActionEvent evt) {
253      //shows a message that tells the reader that they can't go back any more
254      //due to there being no more words
255      if (currentIndex > 0) {
256         //lowers the current position of the word as the user looks back at
257         //previous words
258         currentIndex--;
259         Word.setText(wordArray[currentIndex].getWord());
260         Meaning.setText(wordArray[currentIndex].getMeaning());
261         Example.setText(wordArray[currentIndex].getExample());
262         Image.setIcon(new ImageIcon(this.getClass().getResource("/ImageDirectory/" +
263               wordArray[currentIndex].getImgName())));
264         if (phoneticLanguage) {
265            Pronunciation.setText(((WordWPron) wordArray[currentIndex]).getPronunciation());
266         } else {
267            Pronunciation.setText("Not Available");
268         }
269      } else {
270         //Mastery Factor: Use of additional Java Library
271         JOptionPane.showMessageDialog(null, "There are no more words that come before this word");
272      }
273   }
274
275   //shows the next word in the vocabulary set
276   private void NextActionPerformed(java.awt.event.ActionEvent evt) {
277      //shows a message that tells the reader that there are no more words in
278      //the vocabulary set
279      if (currentIndex < wordArray.length - 1) {
280         currentIndex++;
```

```
281         Word.setText(wordArray[currentIndex].getWord());
282         Meaning.setText(wordArray[currentIndex].getMeaning());
283         Example.setText(wordArray[currentIndex].getExample());
284         Image.setIcon(new ImageIcon(this.getClass().getResource("/ImageDirectory/" +
285               wordArray[currentIndex].getImgName())));
286         if (phoneticLanguage) {
287            Pronunciation.setText(((WordWPron) wordArray[currentIndex]).getPronunciation());
288         } else {
289            Pronunciation.setText("Not Available");
290         }
291      } else {
292         JOptionPane.showMessageDialog(null, "There are no more words that come after this word");
293      }
294   }
295
296   //exits the view form
297   private void ExitActionPerformed(java.awt.event.ActionEvent evt) {
298      this.dispose();
299   }
300   /**
301    * @param args the command line arguments
302    */
303   // Variables declaration - do not modify
304   private javax.swing.JTextField Example;
305   private javax.swing.JButton Exit;
306   private javax.swing.JLabel Image;
307   private javax.swing.JLabel Label;
308   private javax.swing.JTextField Meaning;
309   private javax.swing.JButton Next;
310   private javax.swing.JButton Previous;
```

```
311     private javax.swing.JTextField Pronunciation;
312     private javax.swing.JLabel VocabSetTitle;
313     private javax.swing.JTextField Word;
314     private javax.swing.JLabel jLabel2;
315     private javax.swing.JLabel jLabel3;
316     private javax.swing.JLabel jLabel4;
317     private javax.swing.JLabel jLabel6;
318     private javax.swing.JSeparator jSeparator2;
319     // End of variables declaration
320 }
```

C2: Error Handling

Inputs	Possible Errors	Outputs
Edit a vocabulary set	1 User tries to edit a vocabulary set without selecting a vocabulary set to edit 2 User tries to edit Words Not Yet Memorized Set	1 JOptionPane shows messag “Please select a vocabulary se to edit”(EditActionPerformed method, StartScreenForm) 2 JOptionPane shows messag "Cannot edit this vocabulary set." (EditActionPerformed method, StartScreenForm)
View a vocabulary set	User tries to view a vocabulary without selecting a vocabulary set	JOptionPane shows message "Please select a vocabulary se to view" (ViewActionPerformed, StartScreenForm)
Test a vocabulary set	User tries to test a vocabulary set without selecting a vocabulary set	JOptionPane shows message "Please select a vocabulary se to test yourself with" (TestActionPerformed, StartScreenForm)
Delete a vocabulary set	1 User tries to delete a vocabulary set without selecting a vocabulary set 2 User tries to delete Words Not Yet Memorized Set	1 JOptionPane shows messag "Please select a vocabulary se to delete" (DeleteActionPerformed, StartScreenForm) 2 JOptionPane shows messag "Cannot delete this vocabular set. You must do the test for this set and answer all questions correctly to delete it" (DeleteActionPerformed, StartScreenForm)
Saving a vocabulary set	1 User tries to save a vocabulary set without showing giving a title 2 User tries to save a vocabulary set without words	1 JOptionPane shows messag "Please indicate the title of th vocabulary set" (SaveActionPerformed, EditSetForm) 2 JOptionPane shows messag "You have not entered any words to the set"

		(SaveActionPerformed, EditSetForm)
	3 User writes a title that exceeds that character limit	3 JOptionPane shows message "Please limit the title of your vocabulary set to " + AppLogic.TITLE_RECORD_SIZE + " characters" (SaveActionPerformed, EditSetForm)
Writing/editing a word	1 User tries to writes nothing in the fields	1 See complex selection in EnterActionPerformed, AddWordForm
	2 User writes only white spaces in the fields	2 JOptionPane message shows “At least one of the fields contain only spaces" (EnterActionPerformed, AddWordForm)
	3 User writes an invalid image path name	3 JOptionPane either shows "Please enter a valid image file name. Note: you must first enter image file to the ImageDirectory folder" or "Please enter a valid image file name. Note: the picture must be of file type .jpg, .jpeg, or .gif" (EnterActionPerformed, AddWordForm)
	4 User may write pronunciation for a word that doesn’t require it	4 The pronunciation field is locked when the user is working in French or Spanish
Deleting a word	User tries to delete a word without selecting a word to delete	JOptionPane message shows "You must select a word to delete" (DeleteActionPerformed, AddWordForm)

Other errors (caught using try-catch blocks):

- IOException error: (AppLogic: setWorkingDirectory, deleteTitle, editTitle, writeTitle, readTitles, saveWordSet, numWordsSet, readWordSet;

NodeBinTree: saveWordTree; WordTree: populate, insert) Caught using printStackTrace method

- FileNotFoundException error: (AppLogic: writeTitle, saveWordSet; WordTree: insert; NodeBinTree: saveWordTree) Caught using printStackTrace method
- EOFException error (AppLogic: readTitles) Caught using printStackTrace method
- UnsupportedEncodingException error: (AppLogic: saveWordSet; WordTree: insert; NodeBinTree: saveWordTree) Caught using printStackTrace method

Example implantation of the error handling:
Error handling: reading from file using a buffered writer. Simple input output exception error caught with the try-catch block with the stack trace being printed if anything goes wrong.

```
try {
        reader = new BufferedReader(new FileReader(workingDir + "/" + title +
".txt"));
        while (reader.readLine() != null) {
          lines++;
        }
        reader.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
```

Error handling: writing to file using a buffered writer. The try-catch blocks first check to see whether the COMPILER SUPPORTS THE ENCODING OF THE FILE. If not the stack trace is printed. Then an input output exception is caught. Then the FileNotFoundException is caught.

```
Try{
        Try{
                try {
                        out = new BufferedWriter(new OutputStreamWriter(new
                        FileOutputStream(workingDir + "/" + title + ".txt"), "UTF-8"));
                 } catch (UnsupportedEncodingException ex) {
                        ex.printStackTrace();
                }
         } catch (IOException ex) {
                ex.printStackTrace();
         }
    } catch (FileNotFoundException ex) {
        ex.printStackTrace();
    }
```

Error handling: reading from file using a RandomAccessFile. The first try-catch blocks eofexception is used to make sure that the file has not reached the end the file. If so stack trace is printed. Then an input output exception is caught.

```
try {
        RandomAccessFile raf = new RandomAccessFile(workingDir + "/Word Sets
Titles.txt", "rw");
        int numTitles = (int) (raf.length() / TITLE_RECORD_SIZE);
        if (wordsNotMemorized.isEmpty()) {
           titles = new String[numTitles];
        } else {
           titles = new String[numTitles + 1];
           titles[numTitles] = "Words Not Yet Memorized";
        }
        for (int i = 0; i < numTitles; i++) {
           byte[] titleCharArray = new byte[TITLE_RECORD_SIZE];
           try {
              for (int j = 0; j < TITLE_RECORD_SIZE; j++) {
                 titleCharArray[j] = raf.readByte();
              }
           } catch (EOFException e) {
              e.printStackTrace();
           }
           titles[i] = new String(titleCharArray).trim();
        }
        raf.close();
     } catch (IOException ex) {
        ex.printStackTrace();
     }
```

Error handling: writing to file using a RandomAccessFile. The try-catch blocks simply catches the input output exception and the FileNotFoundException by printing the stack trace.

```
     try {
        RandomAccessFile raf = new RandomAccessFile(workingDir + "/Word Sets
Titles.txt", "rw");
        try {
           if (raf.length() == 0) {
              raf.writeBytes(title);
           } else {
              raf.seek(raf.length() + ((TITLE_RECORD_SIZE - raf.length()) %
TITLE_RECORD_SIZE));
              raf.writeBytes(title);
           }
           for (int i = 0; i < (TITLE_RECORD_SIZE - title.length()); i++) {
```

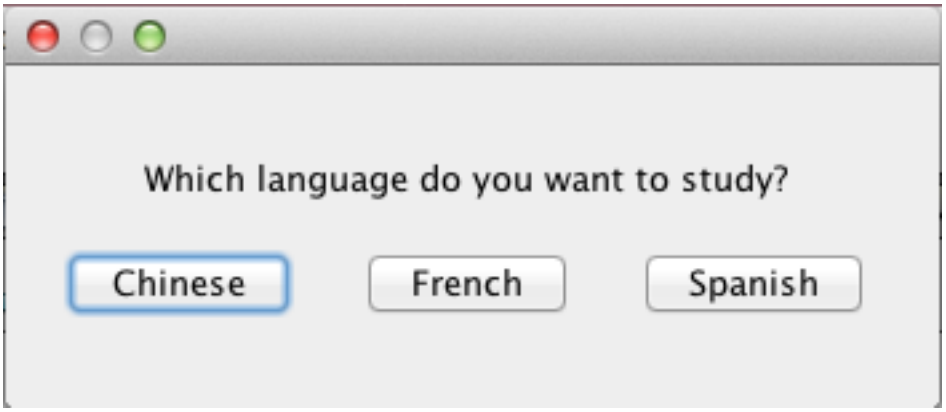
```
            raf.writeBytes(" ");
        }
        raf.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
} catch (FileNotFoundException ex) {
    ex.printStackTrace();
}
```

C3: Success of Program

Goals	Evidence
The program must be self-explanatory—the functions of the program must be laid out so the user knows what each function does without asking questions.	The user is guided throughout the program with messages. Screen 1 forces the user to choose which language they want to go to. Screen 3-6 show message output should the user try to do something they are not allowed to. This forces them to add a new word set by pressing add. Again messages are shown to force the user to add a word. When the user presses add or edit in the Word Set phase, they are reminded that only pictures less 500 by 500 pixels are allowed
The user should be able to add a picture to each word he/she saves.	Screen 14 illustrates that there is a required field of input for the user to place the image path name for them to create a word. Screen 18 and 19 further elaborates on this process, 19 showing the user how to input. Screen 40 illustrates the photo being shown.
The user should be able to add words in a set that are stored for future use.	Screen 20 and 21 illustrate how words are entered in a set. Screen 36 shows how this is saved in disk for future use, from which they are read using the titles found in the Word Set Titles.txt in screen 34.
The words should include example usage of the words, which the students could edit themselves as they see fit.	Screen 13 shows that there is a required example field that the must be written. Students can easily edit this for future use as shown in screen 23 to 26 using the edit button in the word set phase.
The students should be able to test his self to see whether he has memorized the words and give feedback without the need for the teacher to verify that they are correct.	Screen 42 and 43 illustrate how these word sets are tested. By filling in the blanks for the words. Screen 44 illustrates that feedback is given to the user automatically once done button is pressed. Should the user not get a perfect, solutions will be given as seen in screen 45.
The students should be able to track the progress of the words he memorized (should save the words the user hasn't memorized, based on the test, in a new list of words).	The words not yet memorized are saved temporarily in a binary search tree. It is saved into the file as shown in screen 48. This does not disappear until the user has completed the test for the vocabulary set properly. This is shown in screen 47 and 48 again. This is saved in "Words Not Yet Memorized.txt"

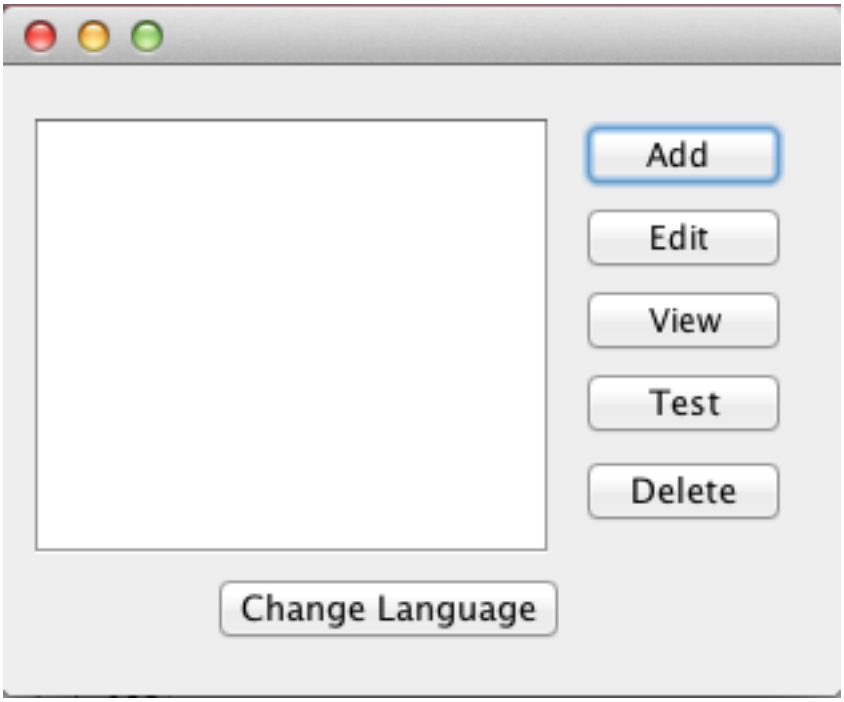
D1: Including an annotated hard copy of the test output

Screen 1



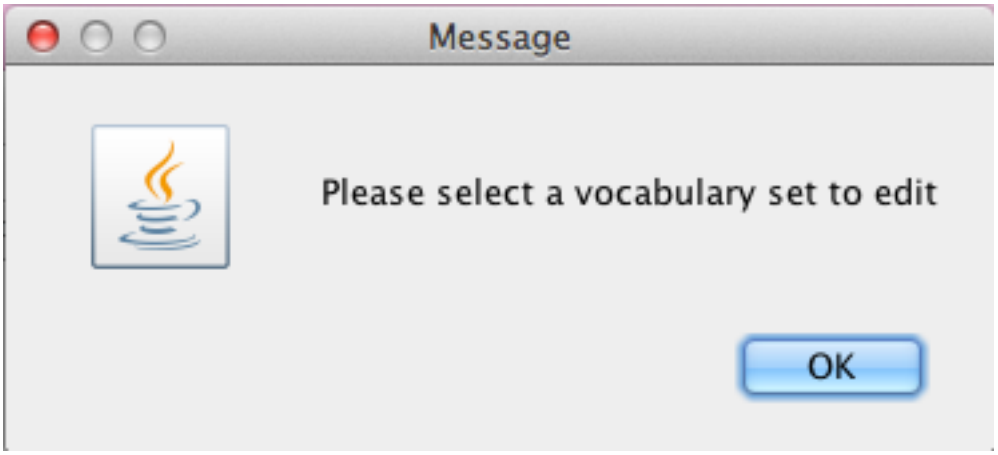
When the program starts, the user is prompted to choose which language they want to study.

Screen 2

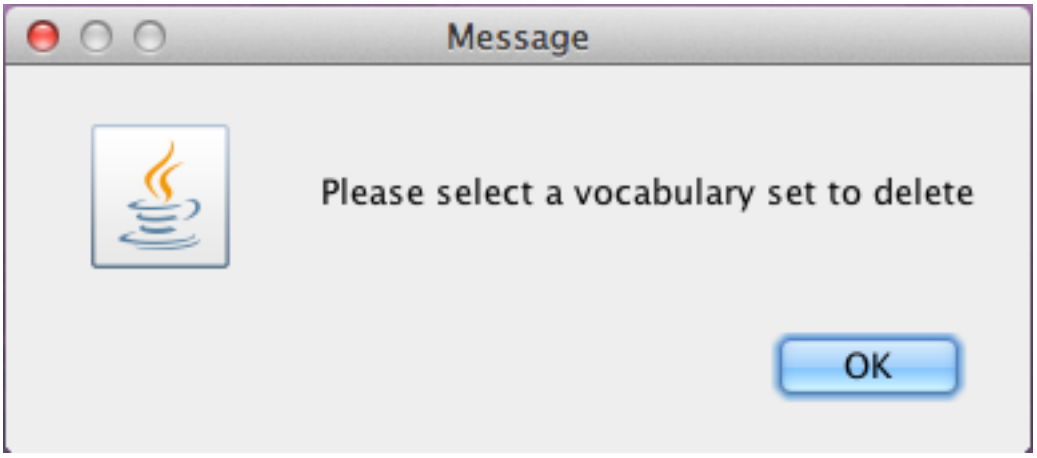


Once they have chosen a language they are sent to a StartScreenForm. Here the only thing that they can do is to add a new word list or change the language. Error handling is carried out to ensure that they cannot perform the functions edit, view, test, delete since there is no vocabulary set to work with. An error message is thrown.

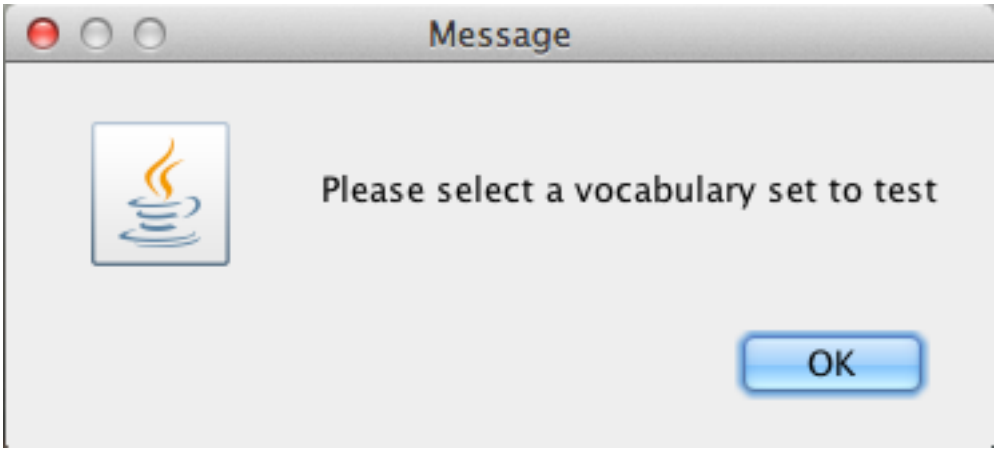
Screen 3:



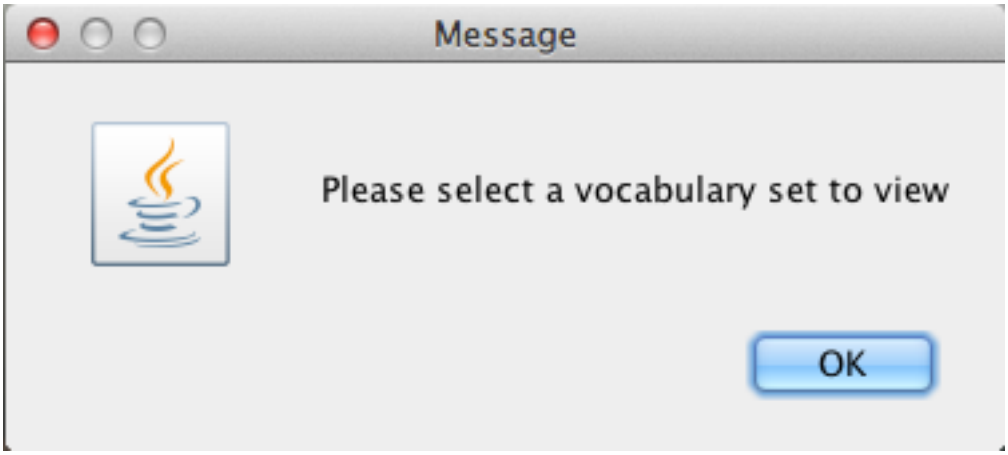
Screen 4:



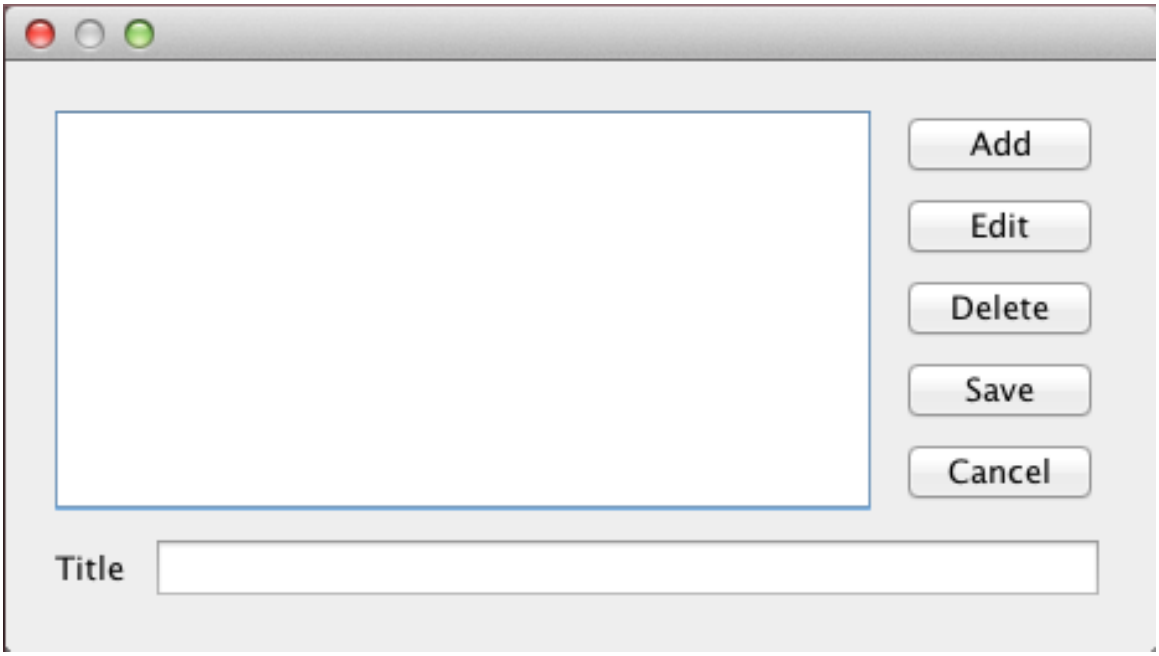
Screen 5:



Screen 6:

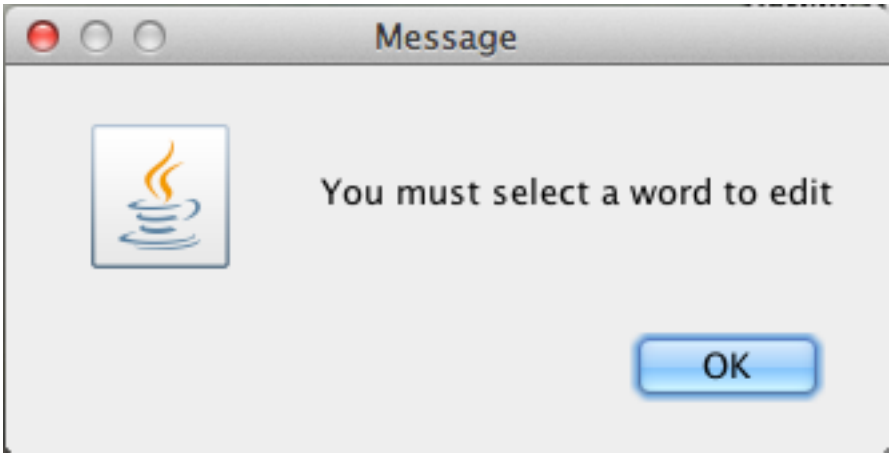


Screen 7:

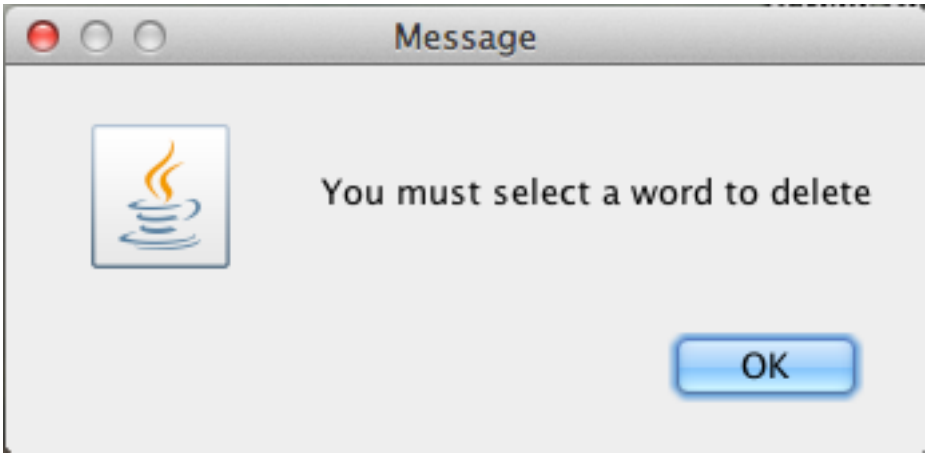


Say the user chooses to **add a new vocabulary set.** Then they click the add button and are taken to this window. They can only add or save at the moment since there are no words in the set and there is no set title. If they press edit, delete or save a message is shown.

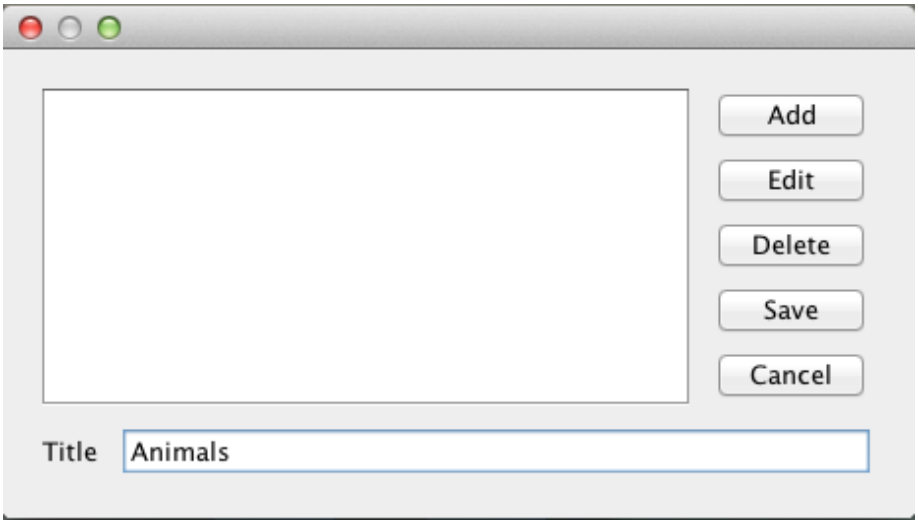
Screen 8:



Screen 9:



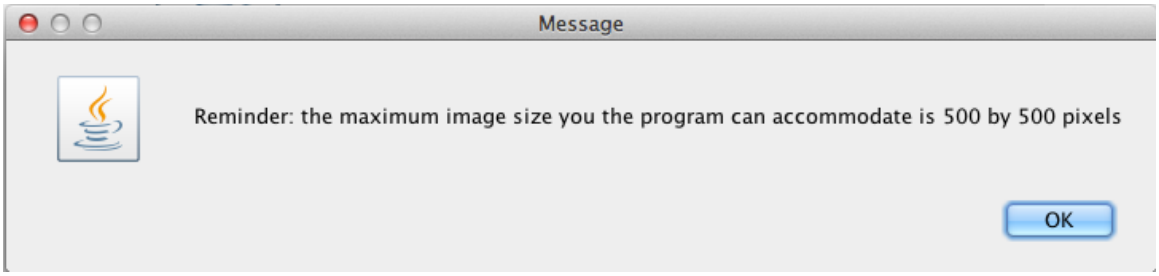
Screen 10:



When the user writes a title and tries to save, a message is shown saying that there are no words in the set.

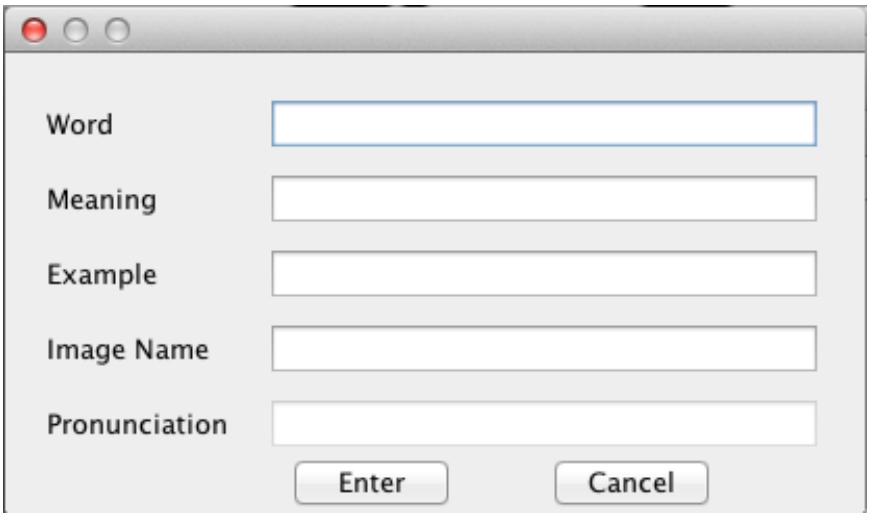
Screen 11:

Screen 12:



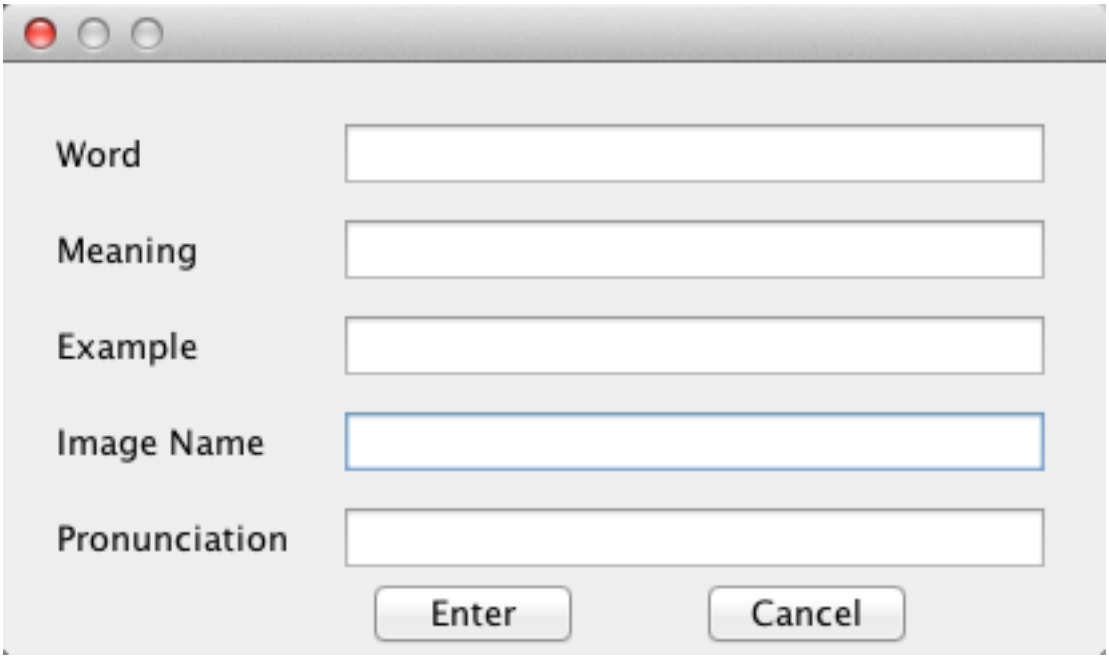
When the user chooses to **add**/edit **button** they are shown this message. This message is a reminder to the user that the images cannot exceed 500 by 500 pixels.

Screen 13:



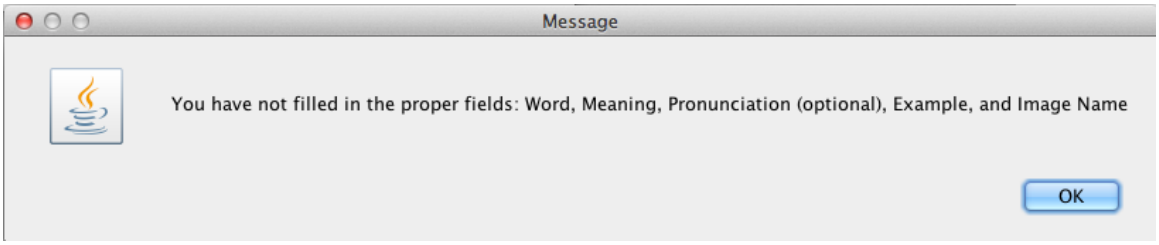
They are then taken to this window where they need to enter the word, meaning, example and proper image name. As shown, the pronunciation field is locked because pronunciation is not applicable to French. In Chinese, as shown below, this is not the case.

Screen 14:

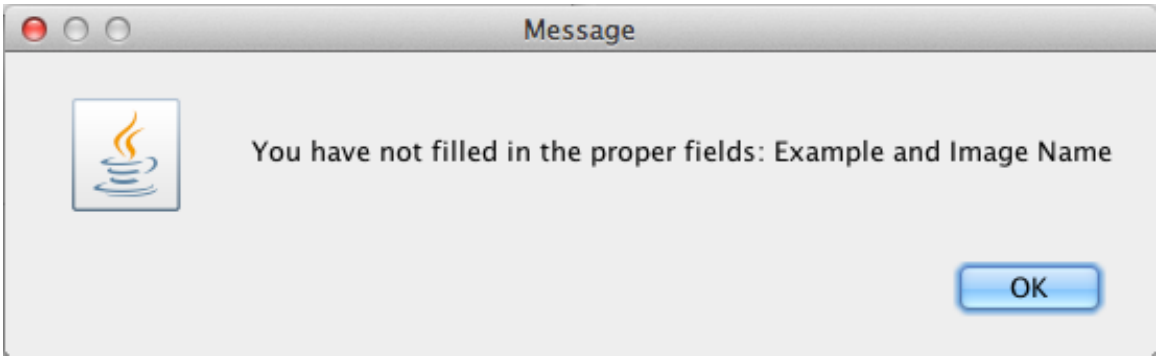


Proper error handling has been accommodated such that the user cannot leave a field blank, or have only spaces in the field.

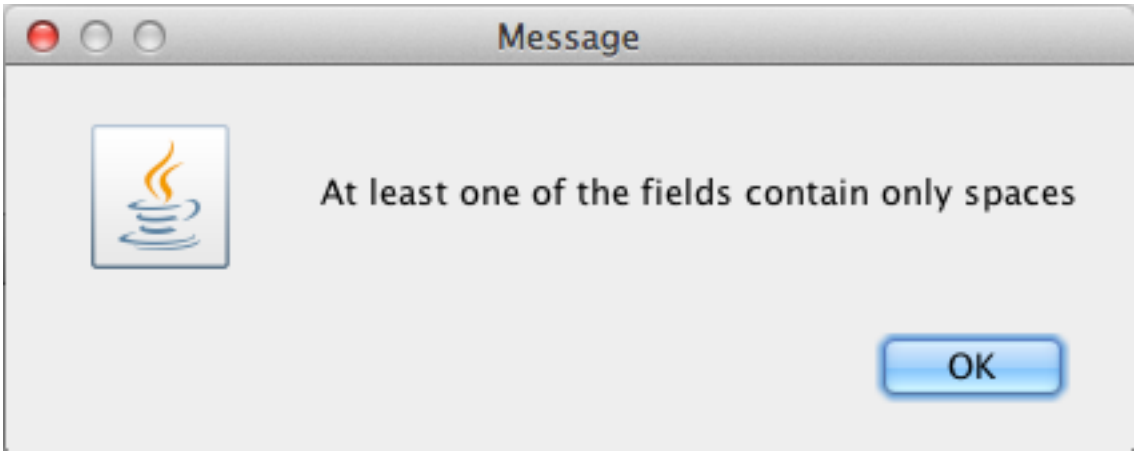
Screen 15:



Screen 16:

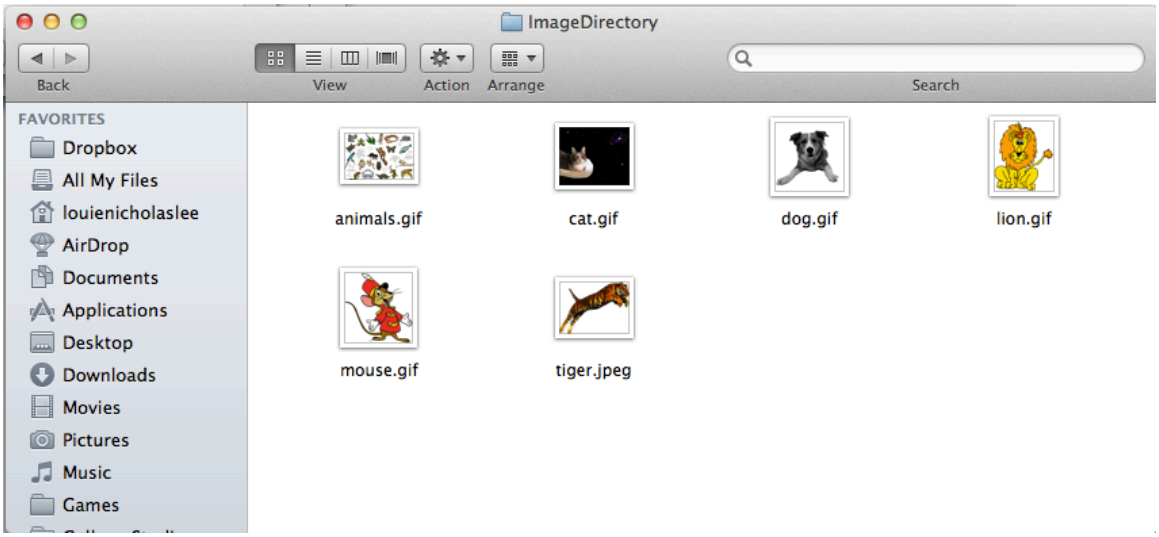


Screen 17:



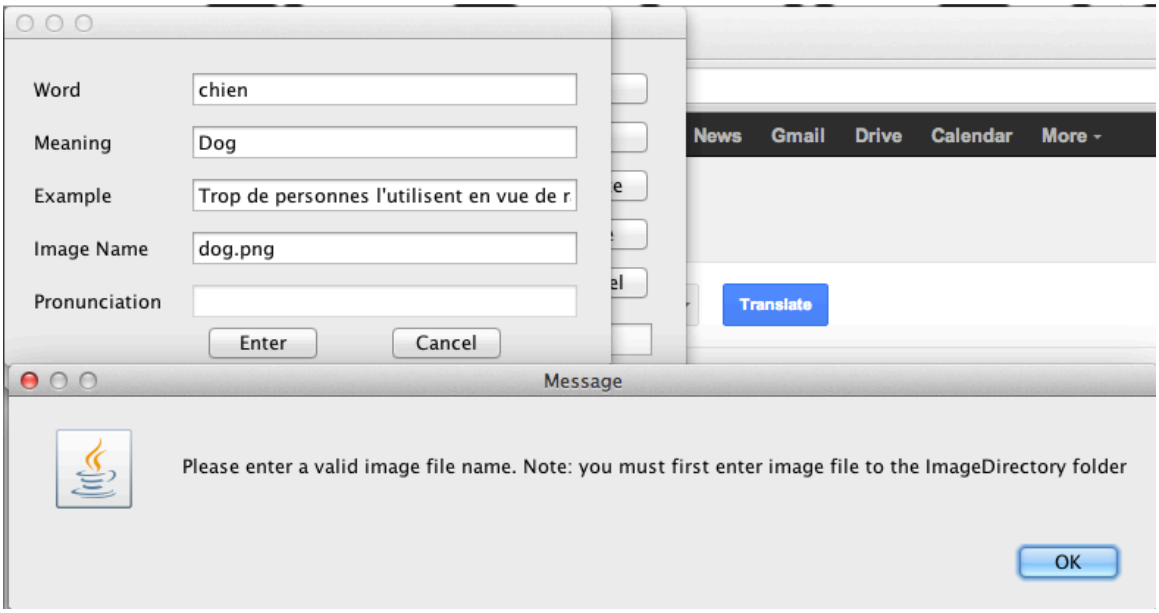
The errors are specified directly to the user so that it is easier for them to track down and change what is wrong, making the program easier to use, a goal in section A2.

Screen 18:

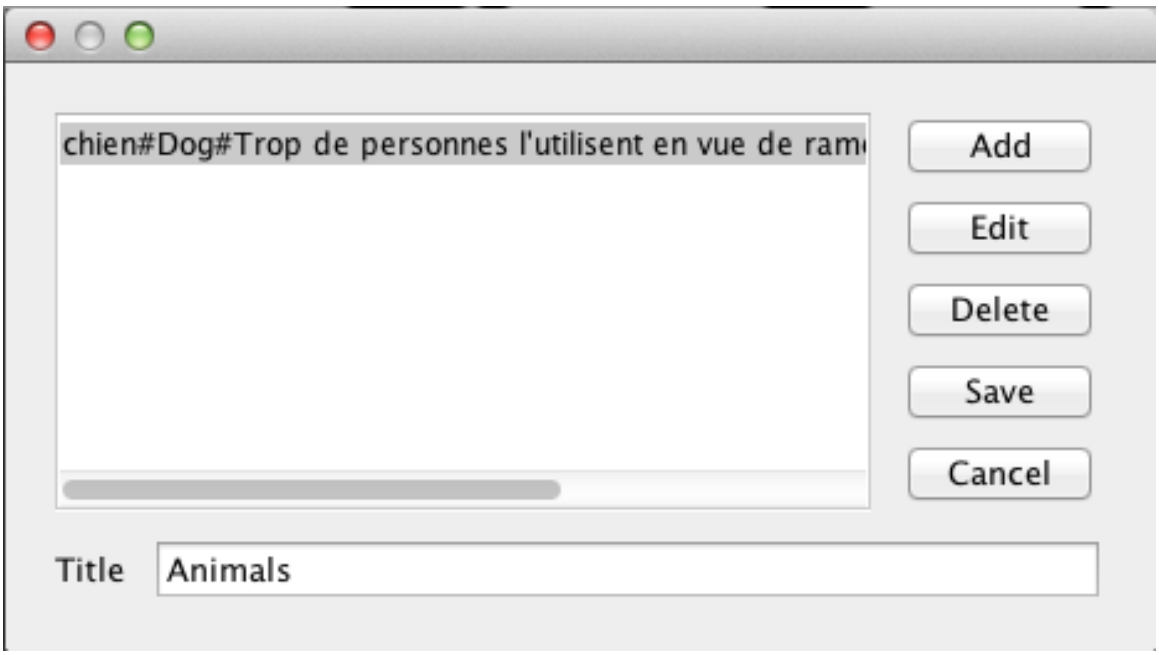


Another error handling taken into consideration is the input of the image path name. The image name entered must be within the image directory and must be of type jpg, jpeg or gif.

Screen 19:

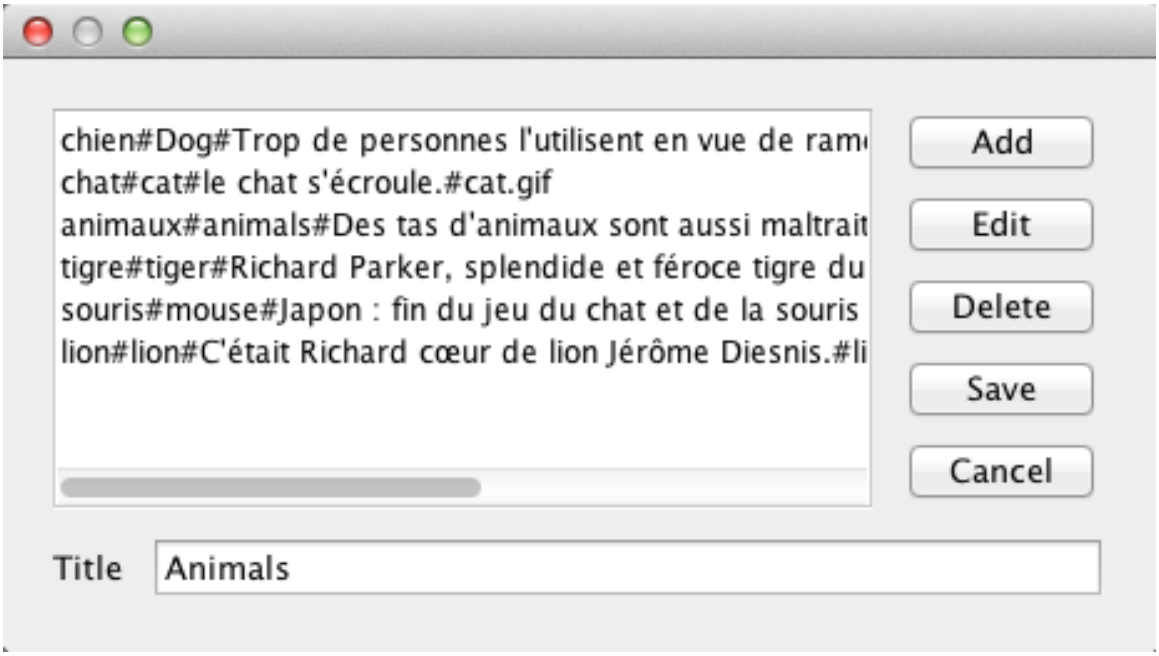


Should the user choose to press cancel, the form just closes and goes back to the add vocabulary set form. When they press enter, the form closes and goes back to the add vocabulary set form with the word entered as shown:
Screen 20:



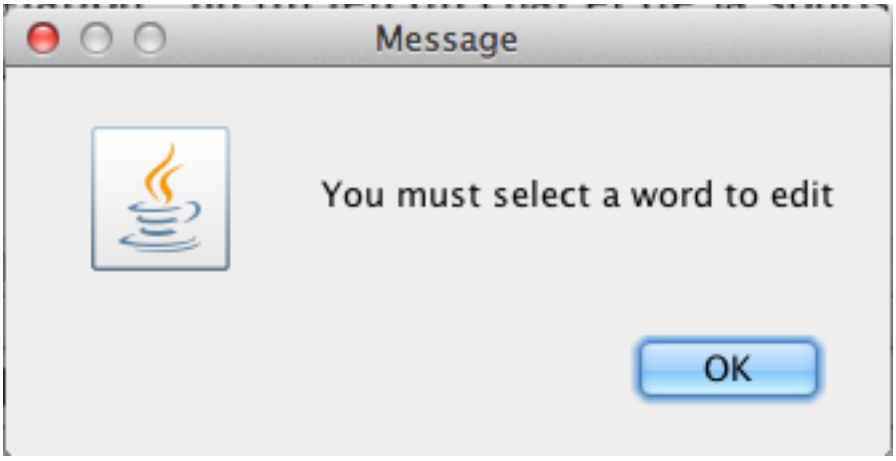
As shown, the separator/delimiter used is hashtag. More words can thus be added in the same manner.

Screen 21:



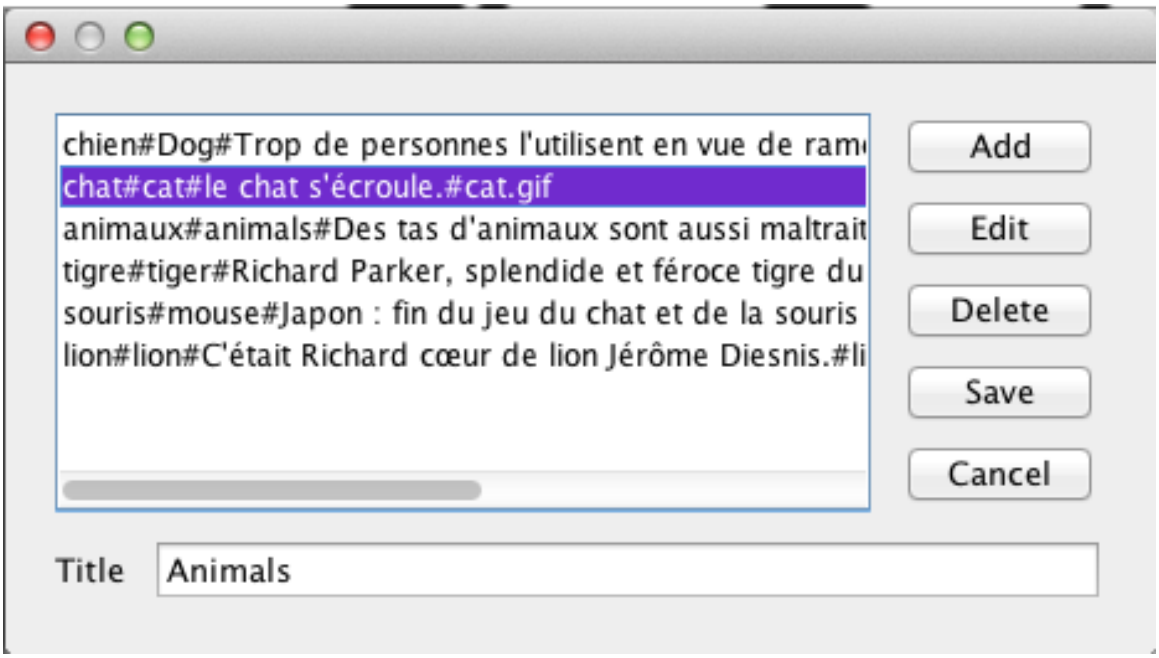
Another function in this form is that the user can **edit** a pre-existing word. However they must first choose to edit one first, or else a message is shown.

Screen 22:

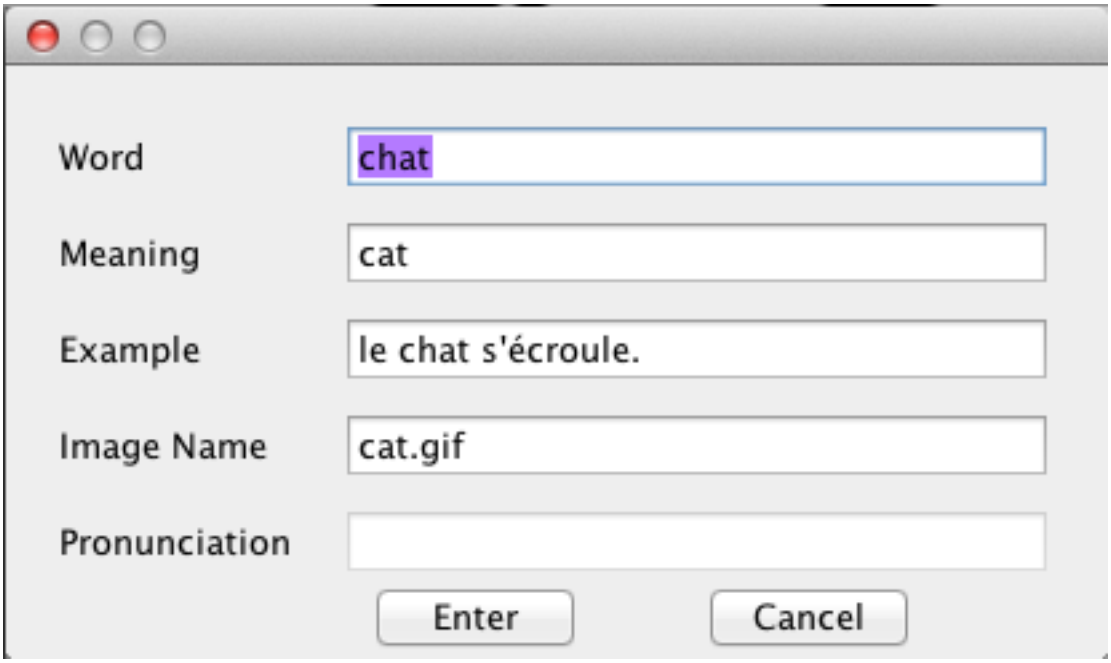


To edit cat, the field is selected and edit button is pressed.

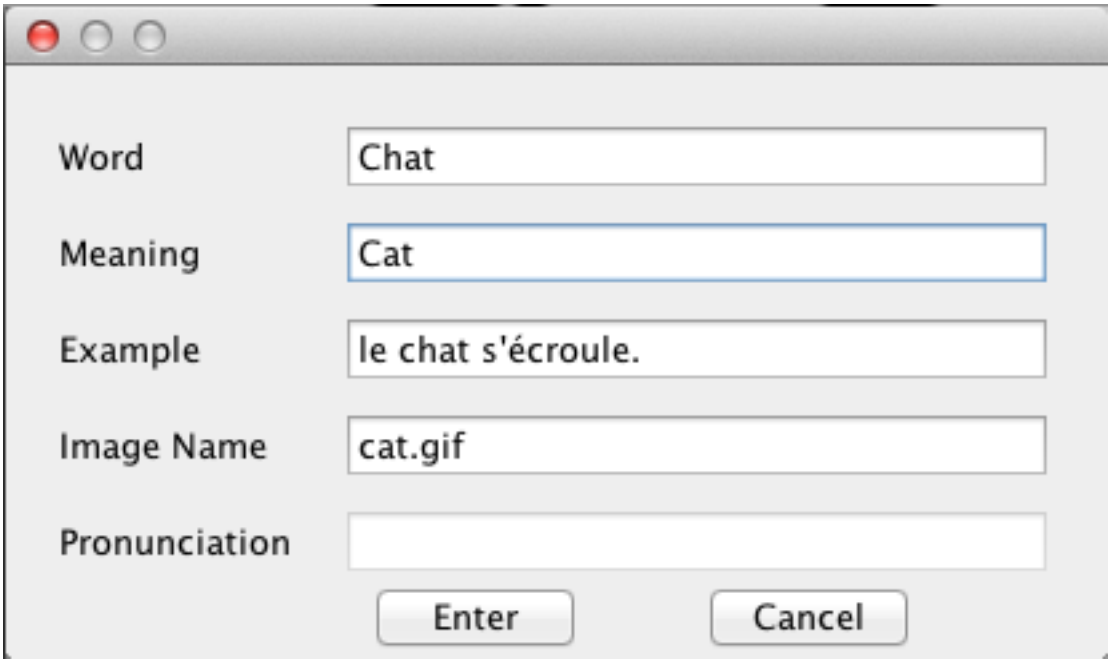
Screen 23:



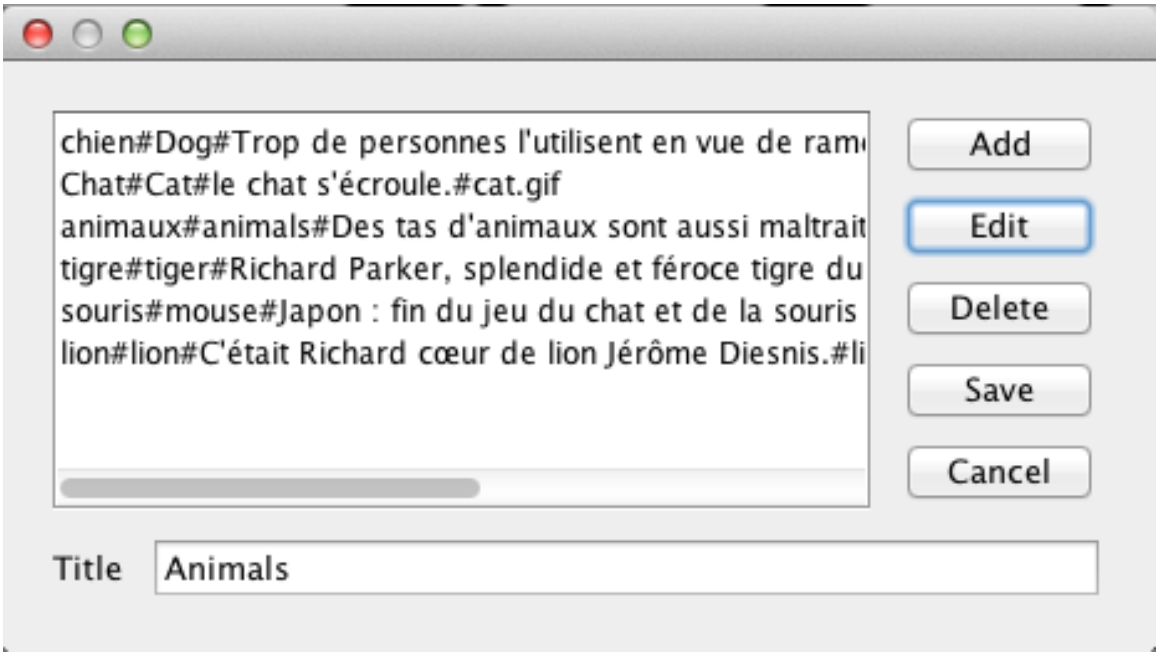
Screen 24:



Screen 25:

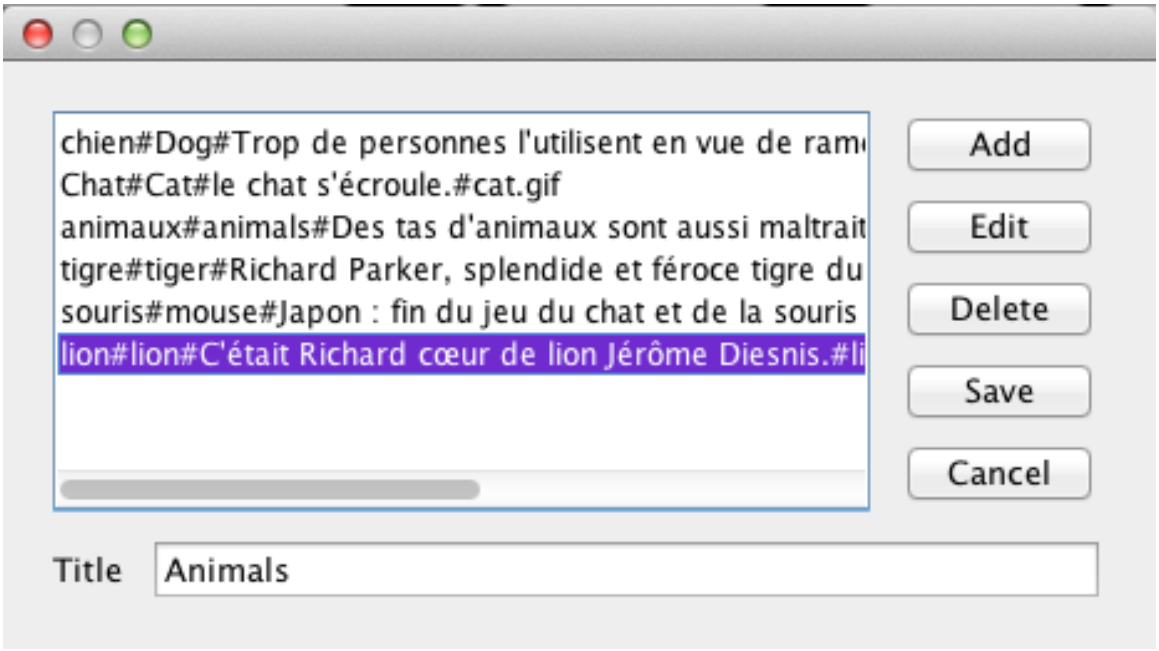


Screen 26:

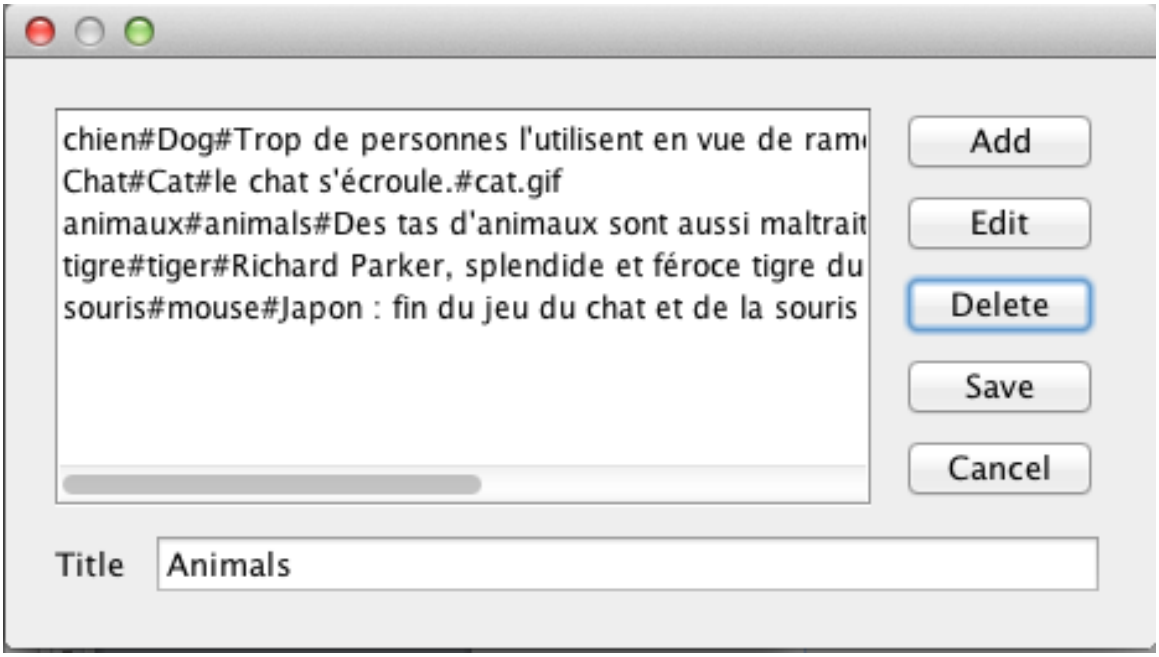


Deleting a word is similar to editing a word.

Screen 27:

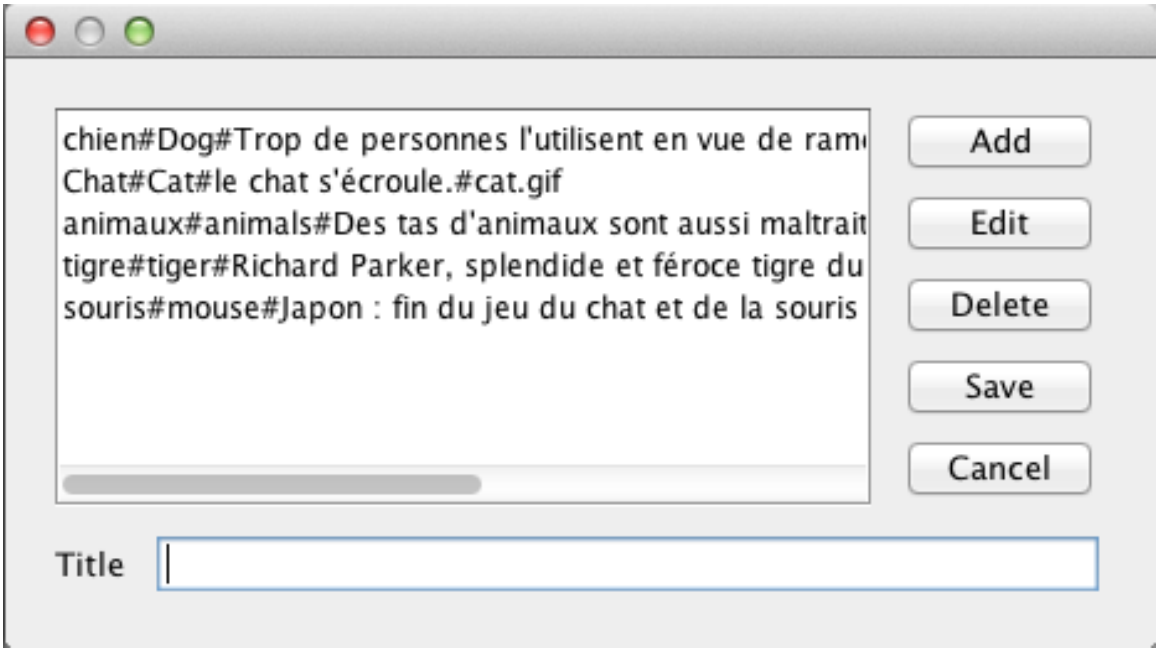


Screen 28:

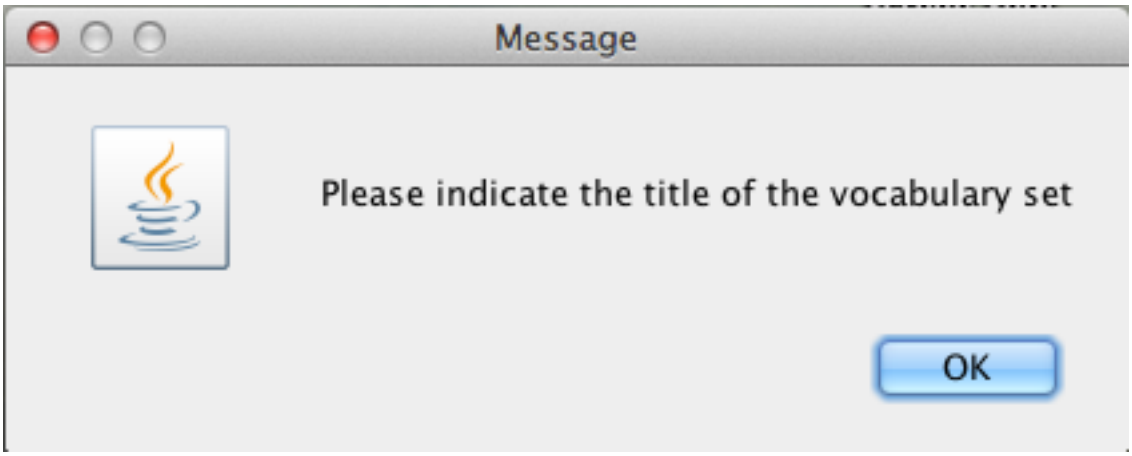


If the user chooses to save without indicating a title,

Screen 29:



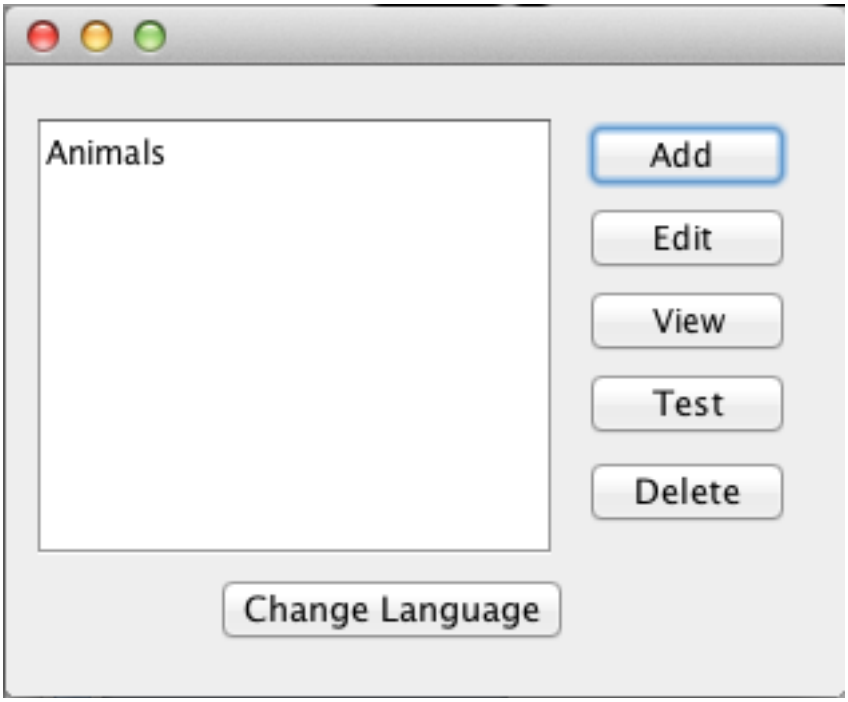
Screen 30:



this message is shown.

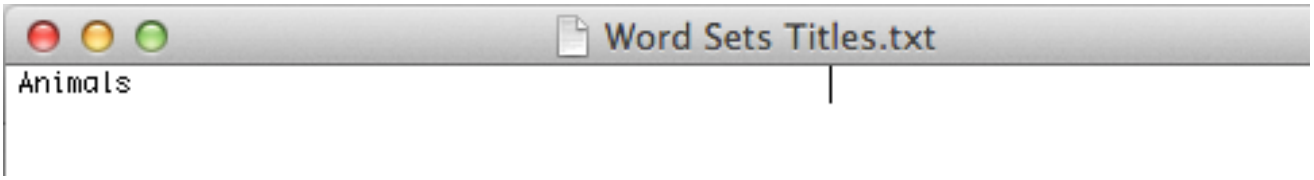
When **saved** properly, the user will be redirected to the start screen with the Animals title shown.

Screen 31:



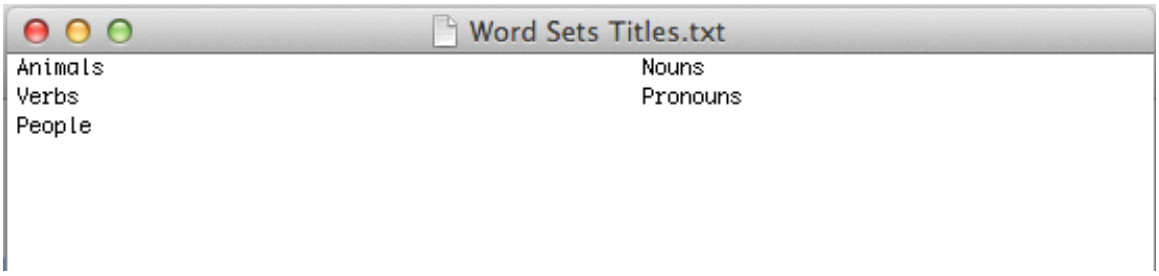
The title is also written to disk in the file “Word Sets Titles.txt” in the corresponding directory (French)
Screen 32:

Screen 33



With more word sets this will look like this:

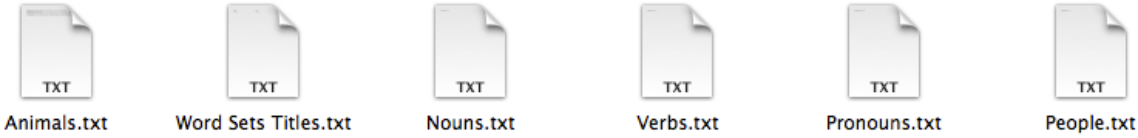
Screen 34:



Evenly spaced out with 50 characters.

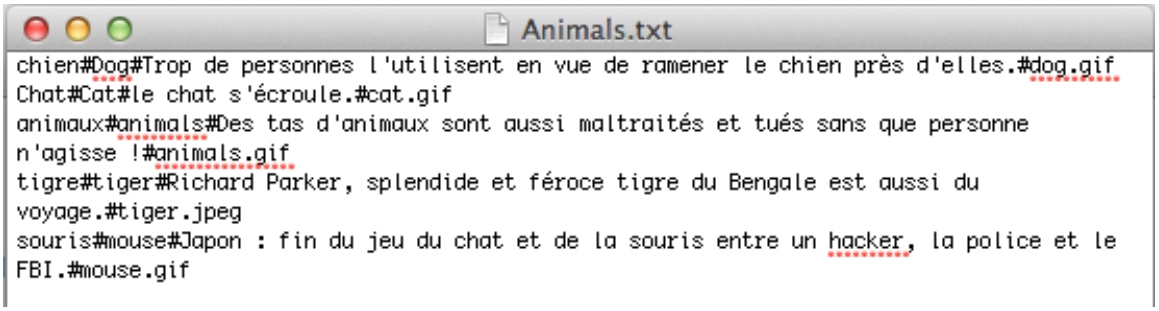
Similarly, when the words are saved onto separate txt files.

Screen 35:



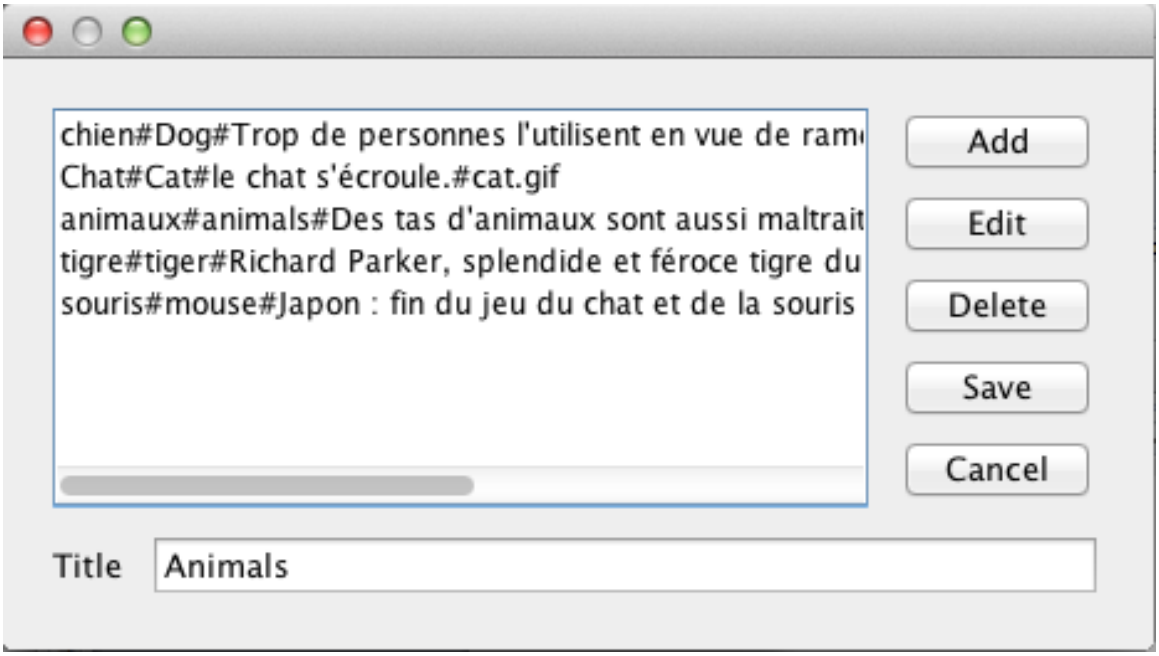
With the previously saved Animals.txt looking like

Screen 36:



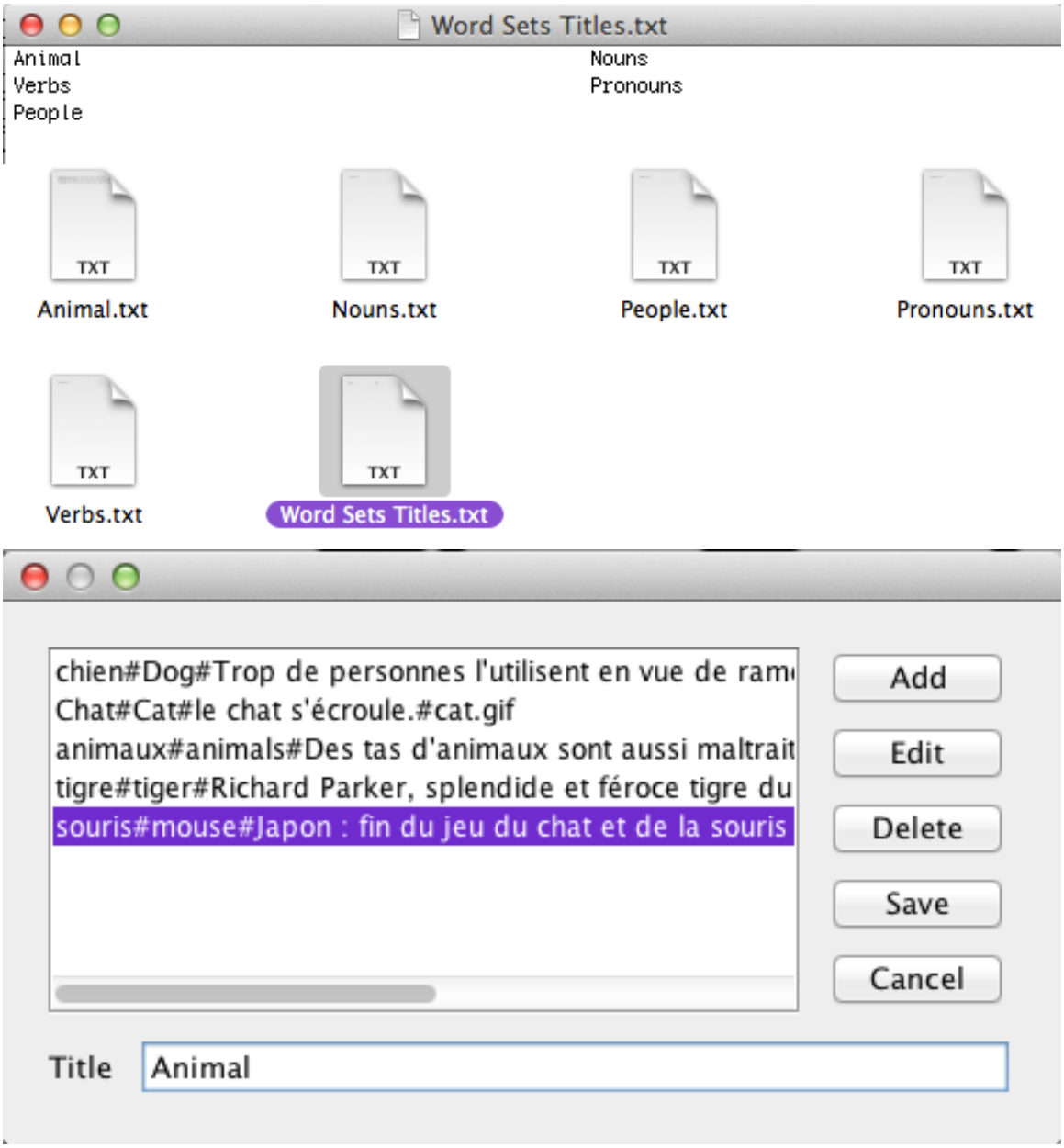
From the start screen, the user can now choose to **edit a vocabulary set**, which brings them back to the add word set form with the corresponding fields already entered

Screen 37:



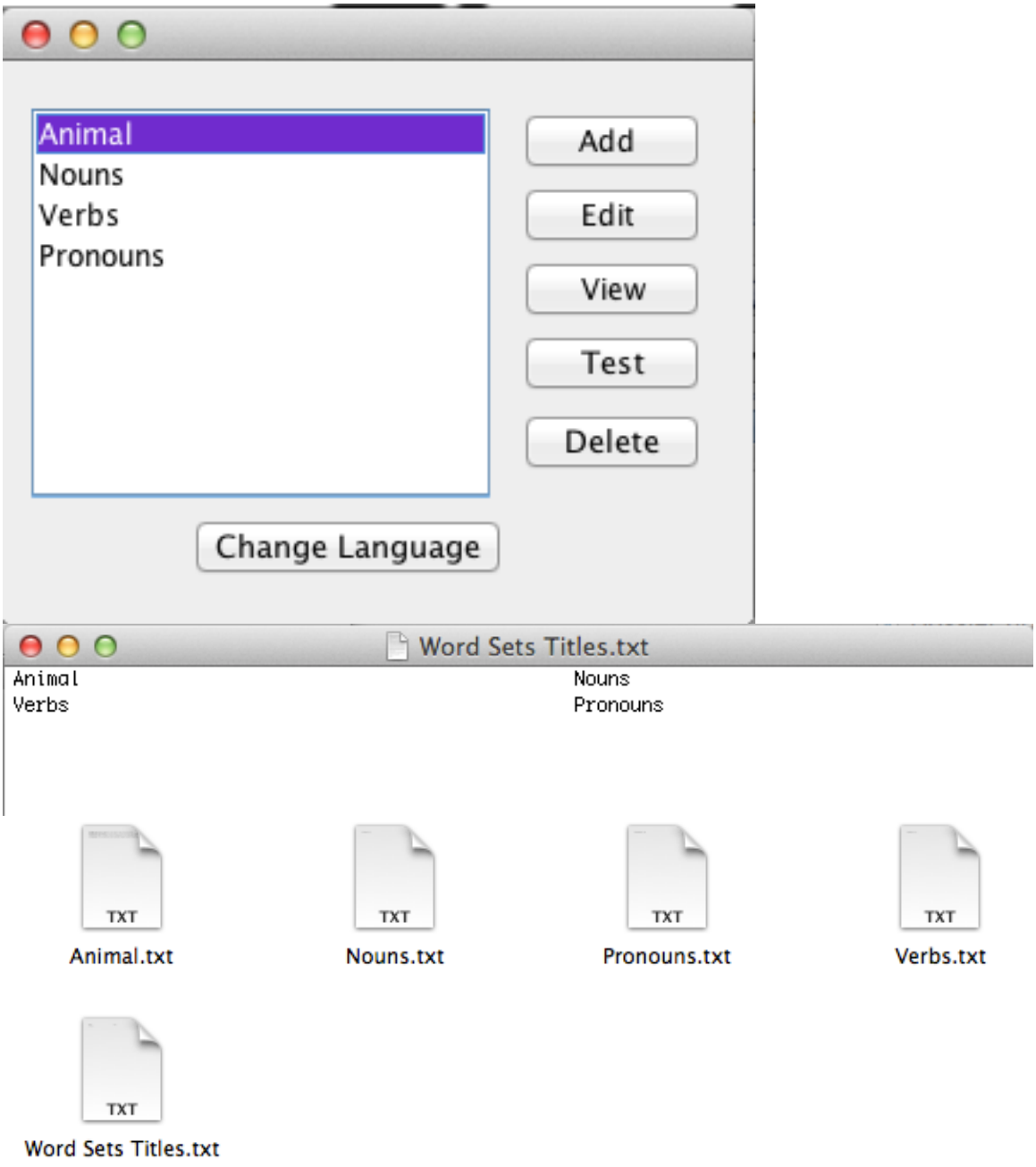
Here they are able to add/edit/delete words as they see fit and change the title of the vocabulary set which automatically adjusts the titles files, and the vocabulary files.

Screen 38:



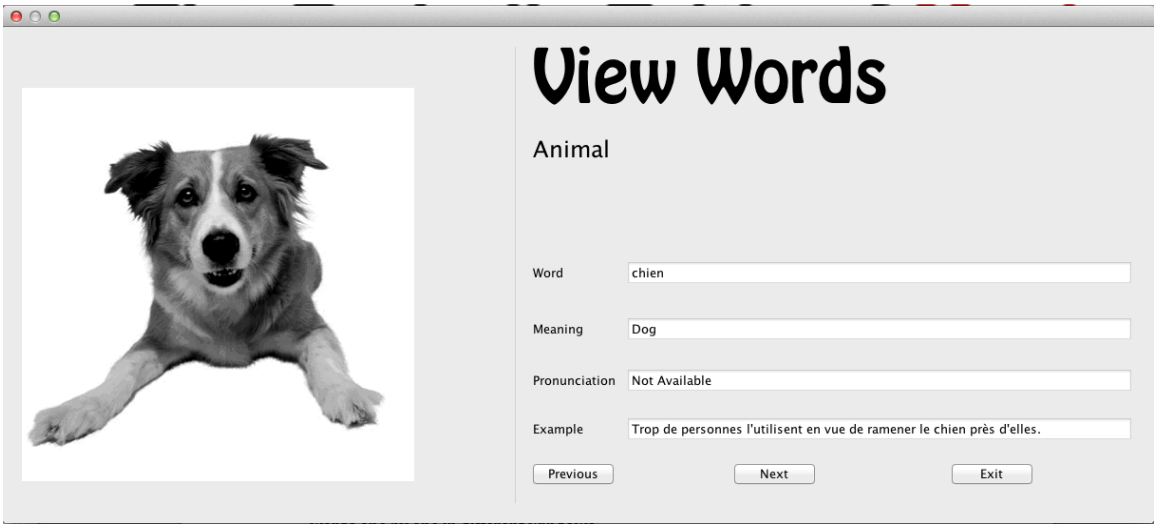
In addition to the edit function, the user can also delete a word list, which automatically deleted its existence from the "Word Set Titles.txt" and its txt file full of words. As shown in the deletion of the "people" vocabulary set,

Screen 39:



Another function in this program is the **view word** list function, which shows the words one by one in different windows:

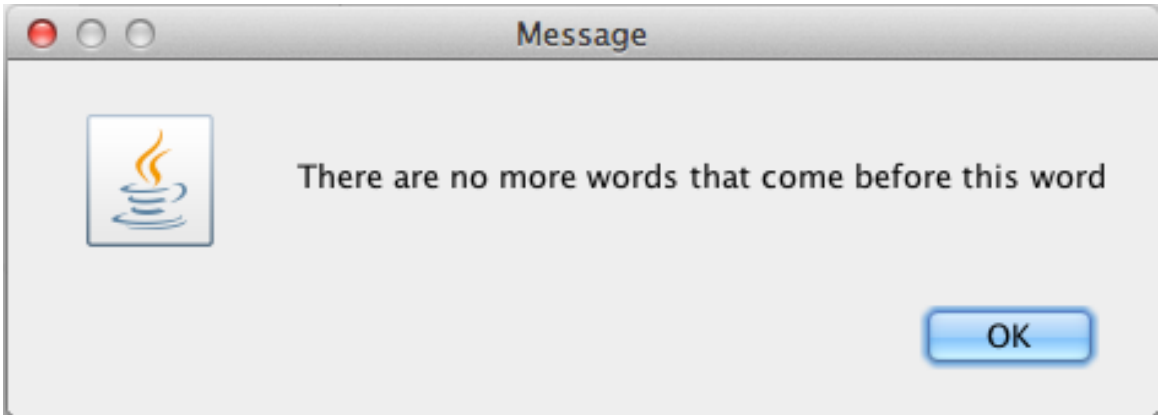
Screen 40:



The pronunciation field is “Not Available” because there are no pronunciation fields for French. The same goes for Spanish, only Chinese has this field filled in.

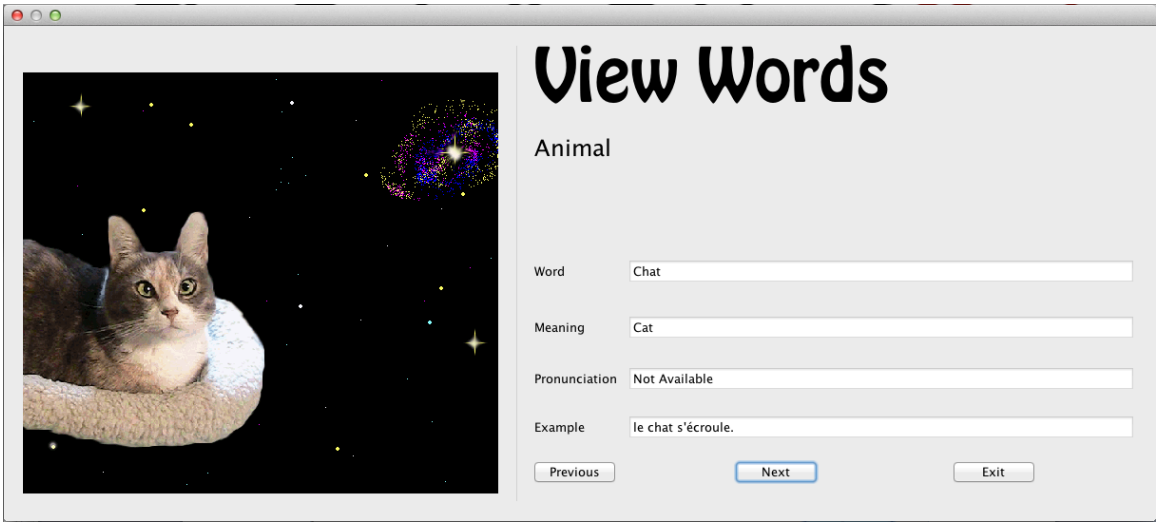
Since this is the start, pressing previous will result in a message.

Screen 41:



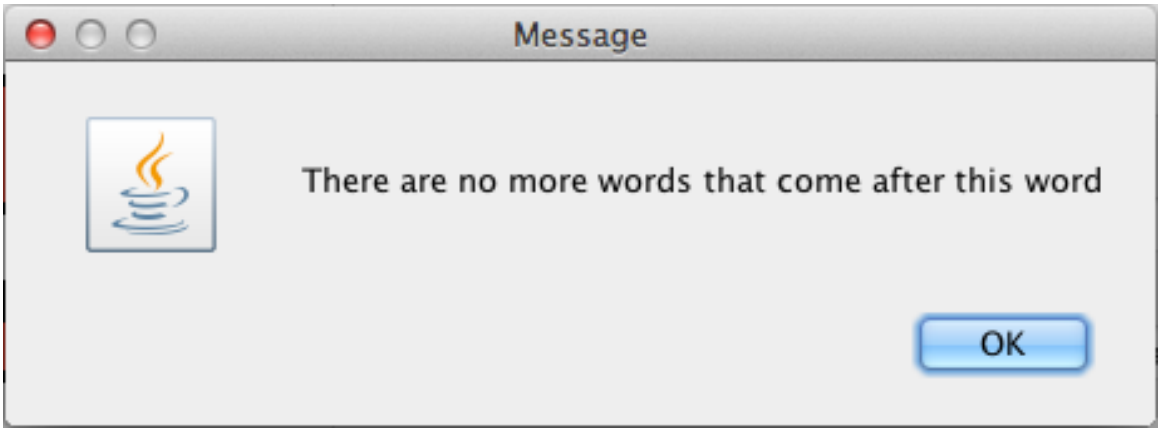
Pressing next will show the next word.

Screen 42:



This goes on until the final word on the set. When pressing next results in the message:

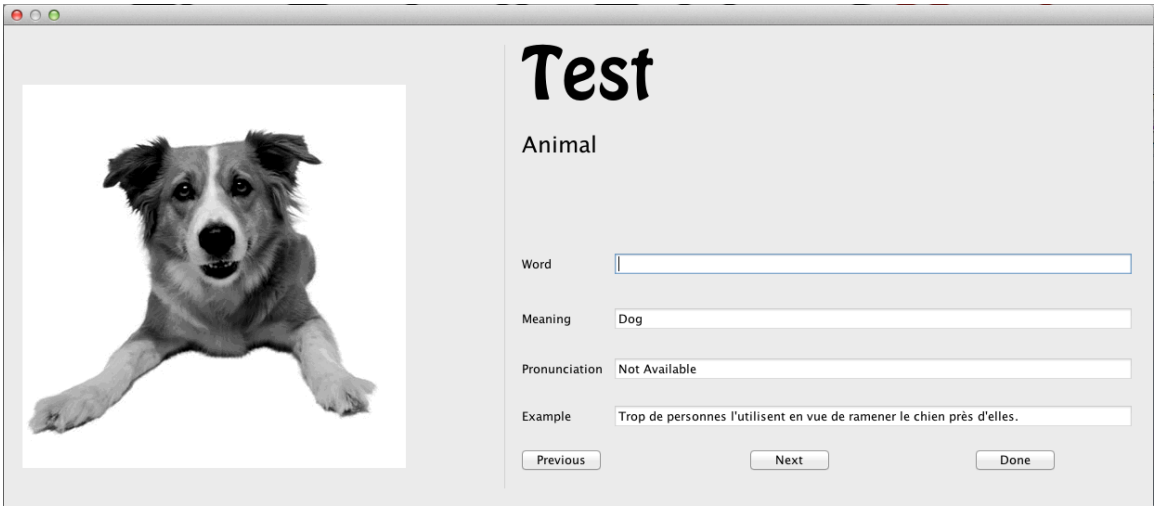
Screen 41:



Pressing exit allows the user to go back to the start screen.

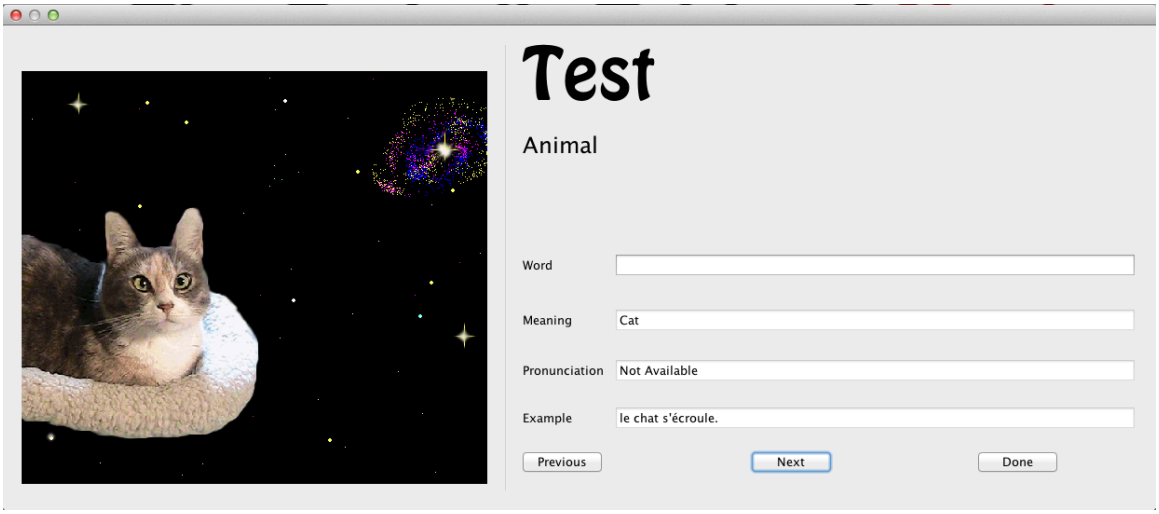
Another function is the **test button**, which allows the user to test their knowledge of the vocabulary sets.

Screen 42:



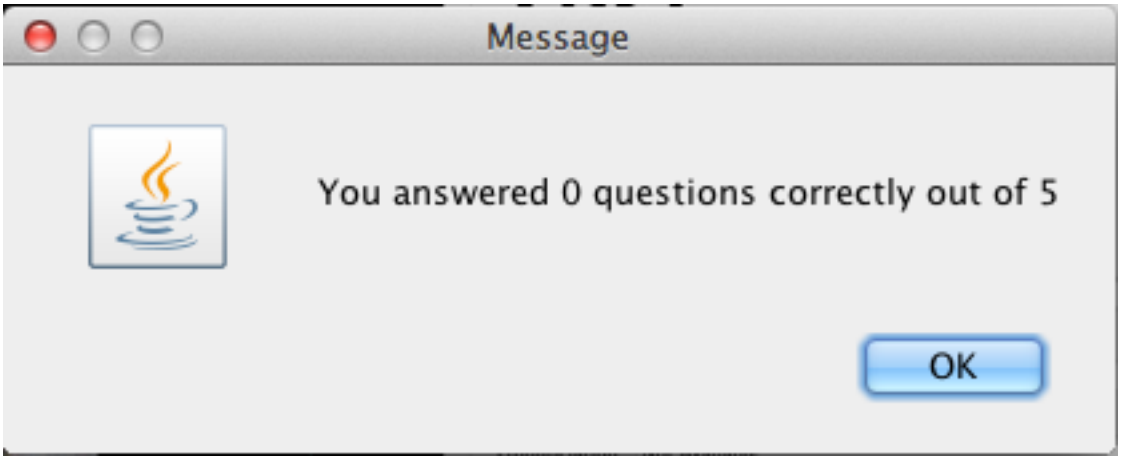
Note that the Word field is blank and editable. This is where the user enters what the word matching the fields is. Clicking next and previous works much in the same way as **view**.

Screen 43



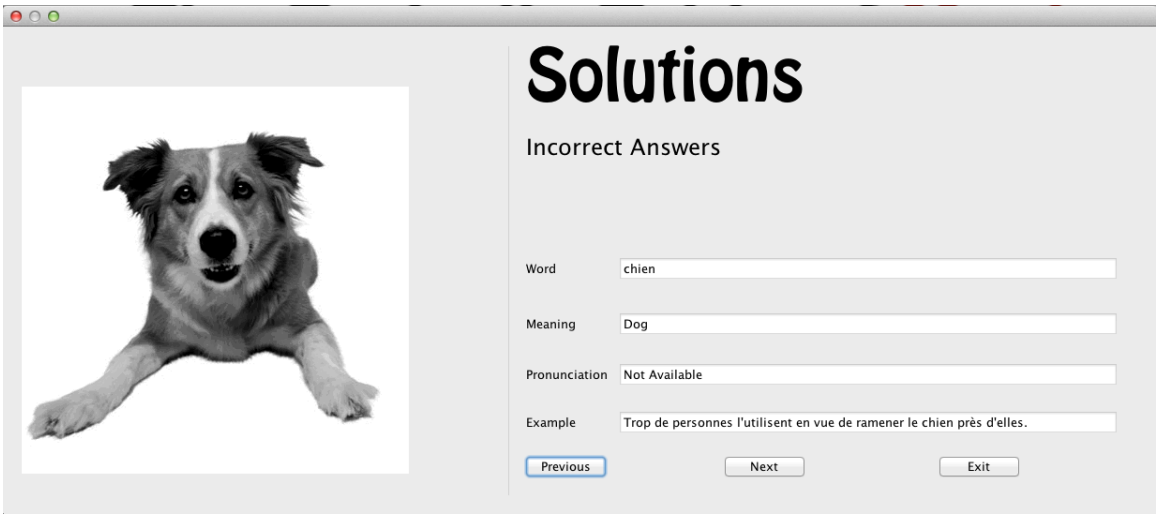
Pressing the done button will result in a message showing the result of the test.

Screen 44:



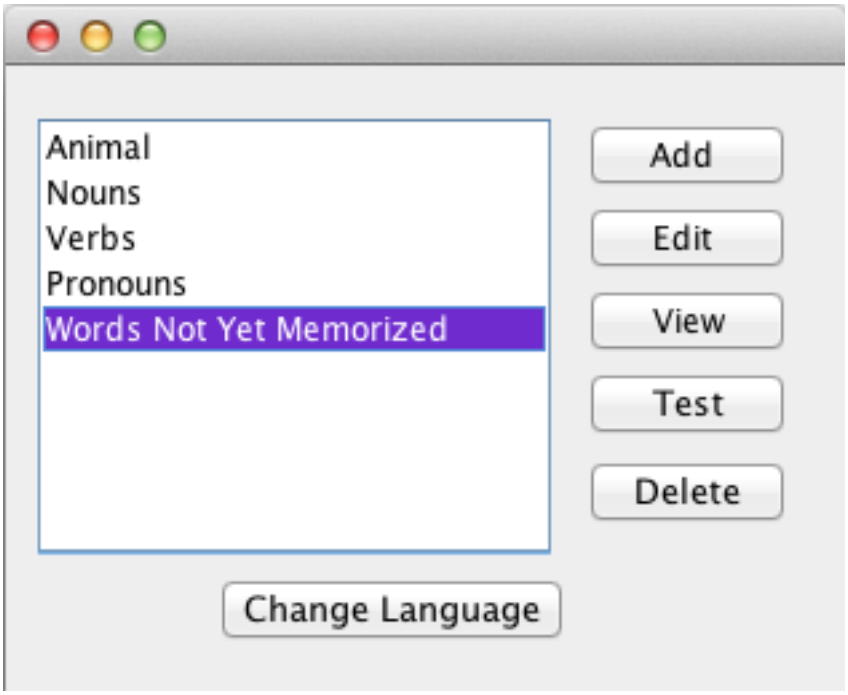
Then the solutions will be shown, if the user does not score a perfect. This is identical to the view form, only that the title is changed to solutions

Screen 45:



Pressing exit leads back to the start screen, as will scoring a perfect score. The difference is that if they don't score a perfect, then a new vocabulary set appears, "Words Not Yet Memorized".

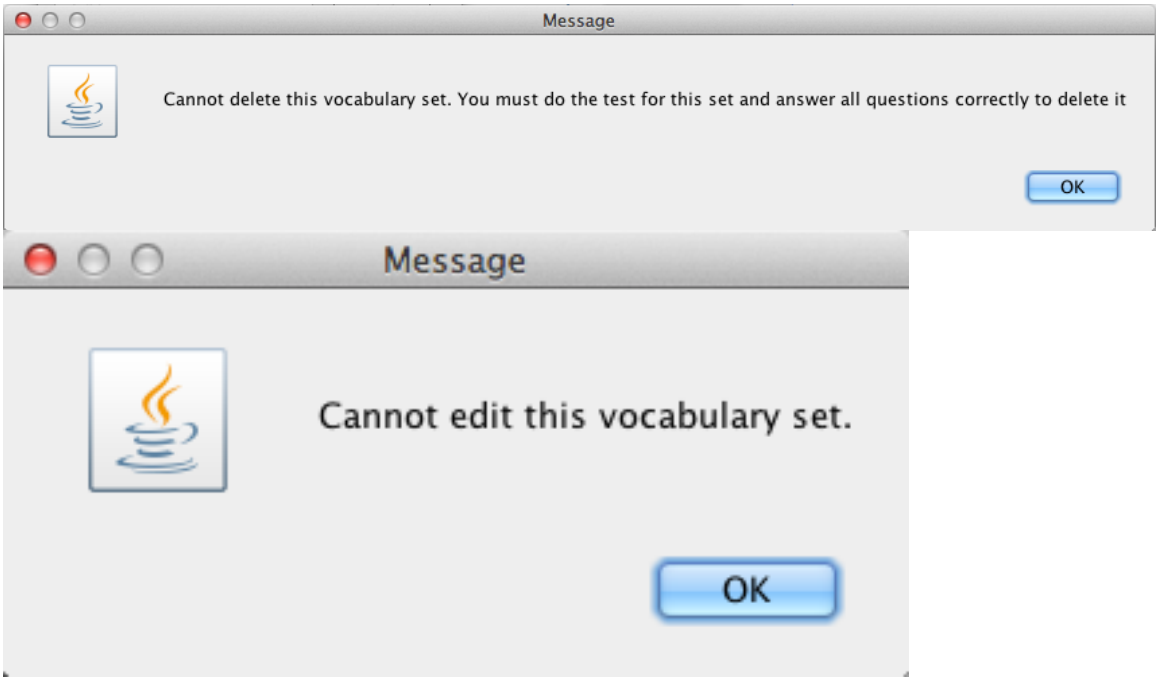
Screen 46:



This shows all the words have not yet been memorized (ones whose questions were incorrectly answered).

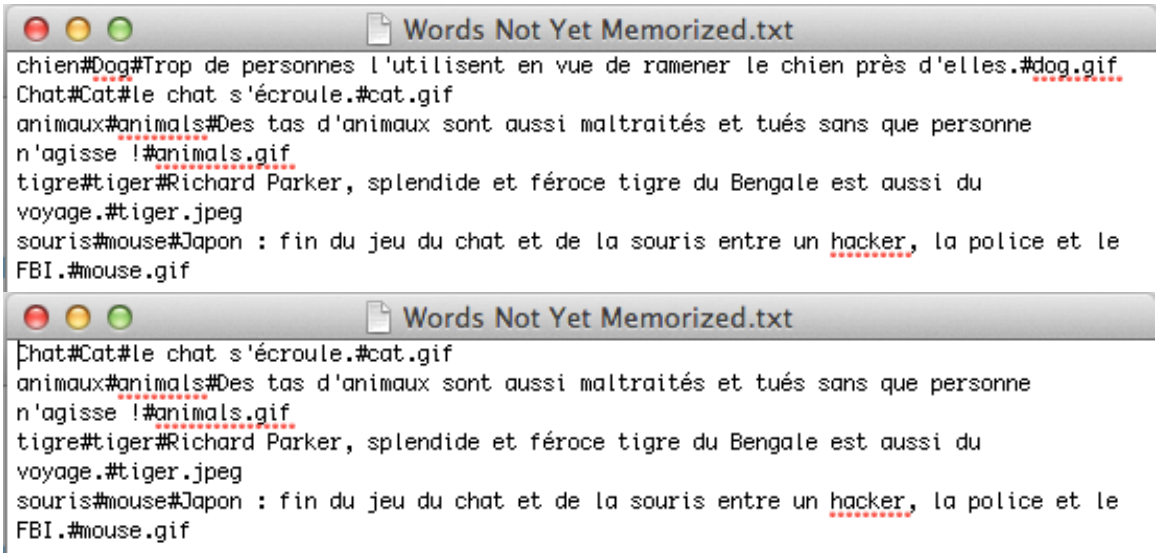
This vocabulary set can be viewed and tested, but not edited and deleted. If one tries, these messages pop up

Screen 47:



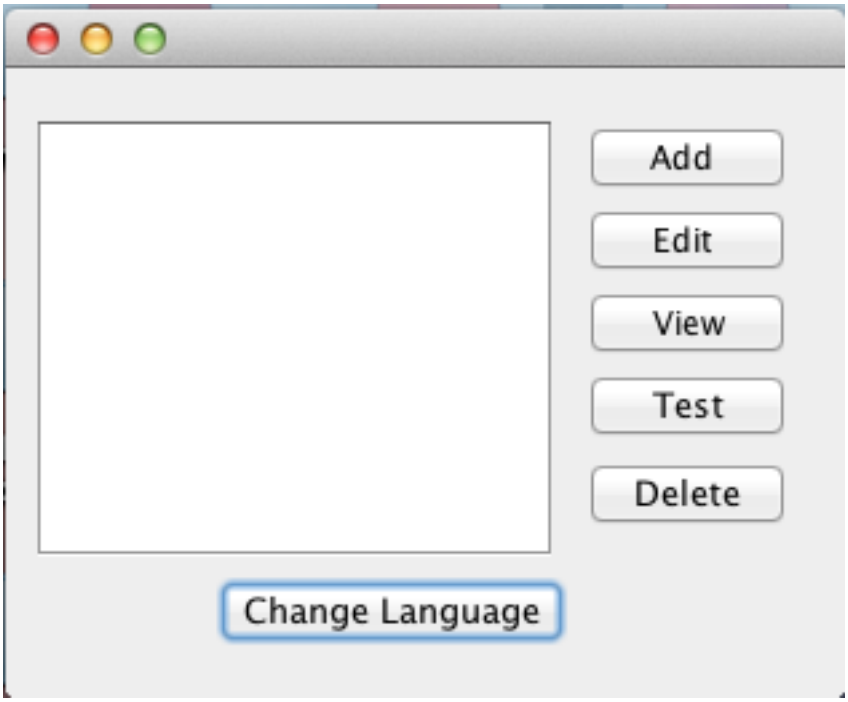
The only way to eliminate this is to take the test and answer the questions correctly. The effect on the txt file will be:

Screen 48:



Changing Languages to Spanish or Chinese will then open the Spanish or Chinese vocabulary sets if they exist.

Screen 49:



D2: Evaluating Solutions

Outline:
The program functions the way it is intended to. It fulfills the basic requirement of being a medium for students to use to improve their study of modern languages by giving them a way to memorize words. There are slight changes to the design such as the addition of the solutions section. The member variables of the Word object are now String word, meaning, example, imgName, and pronunciation. With String word being the word in selected language. Also a hashtag based delimiter is used instead of an asterisk based delimiter because asterisk is not part of the regular regex allowed by the String.split method. However the initial requirements are still fulfilled.
Mr. Pethan's problems as outlined in A1 is to find a way to get students to memorize words without having them keep on bothering him with questions, to check the work of students, to print out copies of words each time a person loses their vocabulary sheet. This computerized solution is stand-alone, so long as there are vocabulary sets that have been made already. The students get automatic feedback and all they need is their laptops (which is available in school).
Effectiveness/Efficiency:
Generally the program is effective in fulfilling the goals and objectives that were set in A2 as shown in C3. The user can utilize the program with relative ease.
Vocabulary sets are saved for future use and can be easily edited. Visual images allow the user to memorize words more efficiently. The view test function of the program utilizes read cover check, pictorial association and rote repetition methods of memorizing words.
Unfortunately, the methods to organize the words in different ways were scrapped from the program because there was little use for it in memorizing words. If anything, it may hinder the memorization because the students will just memorize the pattern and may not recognize the word in other situations.
There are certain limitations to the program. Before a user can add a word, he must find an image and place it in the ImageDirectory. This can be very tedious, but after a while it becomes easier as the user builds up his image directory, he can reuse it. Another limitation is that the images cannot be resized and therefore must be of a certain size (500 by 500 pixels or smaller). This makes it hard find images. In addition, only image types of jpg, jpeg, gif are allowed which further limits the number of images available for use.
It is also very tedious to individually add the words and fill the member fields of the word. But in the long-run, because of the sustainability of the code, these can distributed to Mr. Pethan's different classes each year. While initially tedious, it can eventually be very efficient.
Possible Improvements:

1. The test method should utilize a randomize function to rearrange the words when they appear in the test and when the user views the words. This way the words can be memorized alone without the help of outside cues such as

patterns.

2. The number of different languages should be made variable, with the option of the user creating their own language. In this way, any modern language teacher such as a Korean, Tagalog or Japanese teacher can use this program. This will make the program more versatile and more efficient in the long run.
3. Find a way to export images of different types and to condense large images to smaller ones. This will make the program more efficient and easier to use.

Alternative Solutions:

1. A solution to the problem would be to use Internet databases so the teacher could simply search the database for words that he wants to include. This way the addition of words is much easier and the user only needs to include image name and example usage.
2. Along the same path, images can be saved as URLs instead of files in the image directory. This makes it easier for the user to work around having to download the pictures.

Alternative Approach:

The program was written using a top-down approach. Research was conducted on the problem itself. The design stage came first, looking into the different solutions that existed and the problems that came about with them. The global ideas were considered regarding the problem before breaking it down to smaller components into objectives that must be achieved as shown in A2. Finally culminating into even more modular segments and methods that were used to solve the problem. An alternative approach would have been to use a bottom-up approach, focusing on the methods first and tailoring the solutions to the methods that were written. This is much faster as the functions can be written earlier on in the process and can then be used as a way to solve the problem. This makes it easier to organize the solutions as components easily fit with each other. However a top-down approach is taken because this dossier is intended to focus on how to solve a clear problem.