WESTERN UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SE2205b - Algorithms and Data Structures for Object-Oriented Design

Laboratory 3: Dictionary

Due Date: Match 7th , 2018

Student Name	Jeffrey Lee
Student Number	250894150
Laboratory Date/Time	March 7, 2018/2:30PM-4:30PM

Note:

- **For this lab you will need to submit the full solution (PDF file and code) into both OWL and GitHub.**

1 Goal

In this lab you will use a dictionary in an animated application to highlight words that could be misspelled in a text file.

2 Resources

- Unit 6: Dictionaries
- docs.oracle.com/javase/8/docs/api/—API documentation for the Scanner class
- docs.oracle.com/javase/8/docs/api/—API documentation for the StringTokenizer class
- Dictionay.mp4— The final application demo

3 Java Project Files

- NetBeans project file: Dictionary.zip

4 Create a new repository for Lab3

1. Download the lab3 file SE2205B-Lab3.zip from OWL into your course folder say "SE2205B".
2. Unzip this downloaded file to have a folder called "SE2205B-Lab3". (don't remove the hyphen)
3. Open GitHub Desktop.
4. Run File→ Add local repository.
5. Click Choose and browse for the folder SE2205B-Lab3.
6. You will get a warning says "This directory does not appear to be a Git repository. Would you like to create a repository here instead?
7. Click "create a repository" and then click Create repository.
8. In the GitHub Desktop tool bar, click Publish repository, check the option "keep this code private", choose your GitHub account, and then click Publish repository.
9. Now a new repository called "SE2205B-Lab3" should appear into your GitHub account.
10. It is a mandatory to add the instructor and the TAs to this repository.
    a. In the GitHub account click the repository SE2205B-Lab3 and then click "Setting".
    b. Click "Collaborators & teams" option. At the bottom of the page and in the Collaborators section, enter the account id of the GitHub user your want to add and then click "Add collaborator".
    c. You need to add the following accounts to your repo.
        - **aouda**
        - **kalhazmi2**
        - **rafadaguiar**
        - **rbarboza**

5 Introduction

The dictionary ADT is set of associations between two items: the key and the value. A concrete example is a dictionary, such as the Oxford English Dictionary. It associates words with their definitions. Given a key (word), you can find its value (definition).

There are a number of ways you could implement the dictionary ADT. In this lab a hash table will be used to implement the dictionary. The details of how hash tables work will be considered in depth later in other labs. For now, the important features of a hash table are that they allow fast access and that the items in the hash table are not ordered by their keys.

6 Pre-Lab Visualization

6.1 Loading the Dictionary

Given a dictionary of words and a word to be checked for spelling, what is the key? (What is being searched for?). What is the value? (What do we need to know about the key?)

The key is word to be che ***and the value is*** the word's spelling if found

Reading the file that contains correctly spelled words into the dictionary requires that the file must be parsed (the file must be broken up into pieces each containing a single word). Sometimes the format of the file will be tightly specified. In this case, the format will be pretty loose. The correct words will be in a file. They will be separated by either space or return. Review the API for Scanner, and Java Files I/O.

Write a simple algorithm for reading the words from a file. Assume that the file is already opened as an instance of Scanner with the name ***input***.

Algorithm **loadDictionary**

```
while input.hasNext()
    key = input.Next();
    value = key;
    Add<key, value> to dictionary
```

6.2 Wordlet Class

As the spelling checker executes, it will need to consider each word in the text. As it decides whether a word is spelled correctly or not, that information will need to be associated with the word. The function of the Wordlet class is to remember whether the word is spelled correctly.

The Wordlet class holds a chunk of text (presumed to be a word) and a boolean variable indicating if the word is spelled correctly. Review that class if you have not done so already.

6.3 LinesToDisplay Class

A data structure is needed to hold the lines of wordlets that will be displayed by the animated application. Here are the requirements for the class.

1. It must remember up to 10 lines of checked text.
2. It must know whether words are spelled correctly.
3. It must have a line of text that it is currently composing.
4. It must be able to add a wordlet to the current line.
5. It must be able to move to the next line of text.

Consider the wordlets in a single line. Would an array or a list be preferable for storing them?

An array 

Given that the private state variables of the class are:

```
public static final int LINES = 10;    // Display 10 lines
private ArrayList<Wordlet> [] lines;
private int currentLine;
```

Give an algorithm for adding a wordlet to the current line, if lines[] is already inilized as:

```
lines = (AList<Wordlet>[]) new AList[LINES + 1];
```

Algorithm **addWordlet(Wordlet w)** 

```
lines[currentLine].add(w)
```

Give an algorithm for moving to the next line in the array ***lines***, defined above. It goes to the next line, if the number of lines has exceeded LINES, shift them all up by one.

Algorithm **nextLine ()** 

```
if(currentLine == 9)
    for i = 0 to 8
        clear all words from lines [i]
        assign words from lines[i+1] to words from lines[i]
    clear all words from lines[currentLine]
else
    increment currentLine by 1
```

6.4 Reading Words

While parsing the words in the text file is similar to reading the words from the dictionary, it will turn out that using a StringTokenizer will make the job easier. A double loop will be employed. A Scanner will be used to read the lines and a StringTokenizer will be used to break up each line. Consider the following lines of text.

Just some fun text! (You shouldn't take it seriously.)
Slowly she turned and said, "Bob, I don't know
if I like you any more. Do you know why?"
He replied "As the raven once said: 'Never more'".

What characters in the text indicated the beginning or end of a word (delimiters)?

1st character at the beginning of a line or right after a space indicates the beginning of a word. 1st space right after

Are there other characters that might be delimiters?

The end of line character.

In our application, we would like to mark words that could be incorrectly spelled within a given body of text. This means that we will not be able to just read words (and discard the delimiters), but will need to make sure that every character is in some wordlet.

Review the API for ***StringTokenizer*** and give an algorithm that will read a file and create wordlets. Add each wordlet to the lines display. At the end of every line, go to the next line in the lines display. Do not worry about checking the spelling.

Algorithm

```
while input has input
   read Line from input
   construct a StringTokenizer using LINE
   while a token exists
      get WORD from StringTokenizer
      set variable spellingCorrect to true
      if letter is found from WORD using isLetter()
         spellingCorrect = checkWord(WORD, dictionary)
      construct Wordlet using WORD and spellingCorrect
      call linesToDisplay.addWordLet(Wordlet)
   call showLines(linesToDisplay)
```

6.5 Checking the Spelling

Some of the words that the string tokenizer will produce will be things like "!" and "120". The application will assume that if a word has no alpha characters in it, it should not be marked as incorrect. (An alpha character is a lower or upper case letter.)

Write an algorithm that given a string will return true if all the characters are non alpha characters. The ***isLetter()*** method in the Character Java class can be useful here.

Algorithm

```
boolean result = true
for each character c in a string
    if (character.isLetter(c) == true
        result = false
return result
```

Use **GitHub desktop** to commit your work.

1. Click the changes tab in the left sidebar to see a list of the files that have been changed or added since the last commit.
2. Use the checkboxes to indicate which files should be part of the commit. In this activity, you'll select the "SE2205B Lab3 – Dctionary.pdf" file.
3. Type your commit message (Pre-lab checkpoint) in the Summary field.
4. You will notice that GitHub Desktop has already populated the commit button with the current branch. Simply click the button to commit your changes
5. Click Push origin in the menu bar to push this change to your GitHub account.

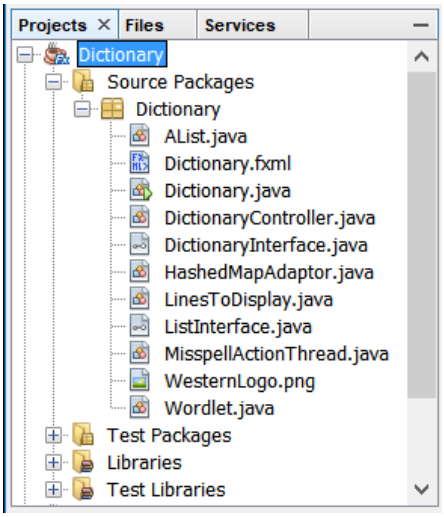
Checkpoint Once you completed the pre-lab visualizations, commit and push this workbook to your GitHub account.

7 Directed Lab Work

7.1 Import Source code to IDE

1. Open NetBeans IDE and import the project "Dictionary.zip". When you import it successfully you will have all files shown in the figure.
2. All required class are given with some empty method that you need to complete as directed in the following sections.

3. At this point, the given should compile and run without error but with incomplete methods.
4. Do not proceed further until you complete this step successfully.



7.2 Startup

All of the classes needed for the MisspellApplication exist. Some of them need to be completed. This application is based on JavaFX GUI interface. If you have not done already, you should look at the code of "DictionaryController", and "Dictionary" classes, and "Dictionary.fxml" file. The classes that you will be completing are LinesToDisplay and MisspellActionThread. The Wordlet class is also specific to this application, but completely defined.

1. Make sure the files named "sampleDictionary.txt" and "check.txt" are in the Dictionary folder (the main project folder).
2. Compile and run the application, from the application window click the button "Check Spelling", the application will do nothing but displays "No Dictionary is loaded".
3. Run and watch the given application demo "Dictionay.mp4' to get familiar with the required output your application will show at the end of this lab activities.

7.3 Loading the Dictionary

1. In the method **loadDictionary()** in **MisspellActionThread**, add code that will read the words and put them into the dictionary. The dictionary file contains words that are either separated by spaces or lines. A single loop is needed. Refer back to the algorithm you wrote for the pre-lab exercises and complete the code.
2. Just after reading in all the words, assign "dictionaryLoaded" to true.
3. In the method **run()**, add a call to **loadDictionary() before the statement:** ***Platform.runLater*()**.
4. Compile and run the application, from the application window click the button "Check Spelling", the application should display "The Dictionary has been loaded".

7.4 Completing LinesToDisplay

1. Complete the constructor for the class **LinesToDisplay** to initialize the private variables.
2. Refer to the pre-lab exercises and complete the code for the method **addWordlet()**.
3. Refer to the pre-lab exercises and complete the code for the method **nextLine().**
4. In the method **run()**, add the following six lines of code

```
myLines.addWordlet(new Wordlet("abc", true));
myLines.nextLine();
showLines(myLines);
myLines.addWordlet(new Wordlet("def", false));
showLines(myLines);
myLines.nextLine();
```

5. Compile and run the application, from the application window click the button "Check Spelling", the application should display the wordlet abc in blue on one line and def should be in red on the next line. If not, debug the code and retest.
6. Comment out the six lines of code entered in the previous step.

7.5 Reading Words

The next goal is to parse the text file into wordlets and put them in the display. For now, even though we call a method to check the spelling, it will always return false. Thus, all words will be considered to have an incorrect spelling.

1. Refer to the pre-lab exercises and complete the code in the method
   checkWords(textFileName, myDictionary).
   As each wordlet is created, use checkWord() to determine if the spelling is correct. (It will always return false until we change that in the next part.
2. **Hint**: make sure you added the call "**showLines(myLines)**;" before moving to next line (i.e., before the statement "**myLines.nextLine**();")
3. Compile and run the application, from the application window click the button "Check Spelling". Each line in the text file should appear exactly with all the text in red.
4. The final goal is to check the spelling of the wordlets.

7.6 Checking the Spelling

1. Complete the **checkWord()** method to search for the word in the dictionary. Return true if the word is in the dictionary.
2. Compile and run the application, from the application window click the button "Check Spelling". Each line in the text file should appear exactly. The words in the dictionary should now appear in blue.
3. It would be nice if the punctuation did not show up in red. Refer to the algorithm from the pre-lab exercises and add code to checkWord() to return true if the word consists only of punctuation characters.
4. Compile and run the application, from the application window click the button "Check Spelling". Most of the punctuation should now be in black.

8 Hand in

For this lab you need to submit the full solution (PDF file and code) into both OWL and GitHub.

1. Using NetBeans IDE, export the Dictionary project to ZIP file and name it yourUwoId_SE2205B_Lab3.zip. Use File → Export Project → to ZIP, then enter the zip file name.
2. Submit at the due date mentioned above
   a. This zip file along with your solution workbook (this PDF file, make sure your answers are saved), using OWL assignment link
   b. Commit and Push your full solution folder along with your solved workbook into your GitHub account.