Worksheet 4: GUI for Predictive Text Entry

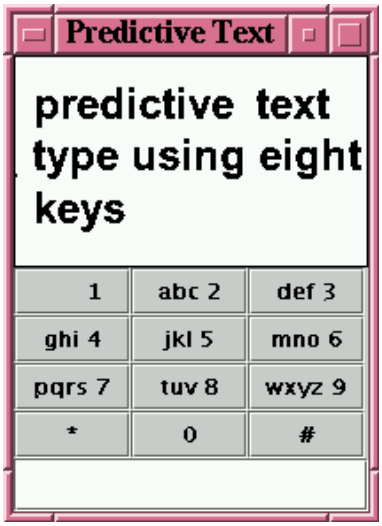
Seyyed Shah and Uday Reddy

Assigned: Tuesday, 14th February
Deadline : Monday, 20th February, 5:00pm

As usual, include in your submission:

1. appropriate comments and JavaDoc.
2. **thorough** testing. (You may use JUnit wherever applicable.)

In this exercise you will develop a a Graphical user interface to the predictive text dictionary implemented in Worksheet 3. The minimal user interface looks like this:



In this GUI, there is one main control mechanism, the buttons in the numeric keypad. The message appears in the text window at the top. By pressing the button "2", the user will get one of "a", "b", or "c". To cycle through the words (or prefixes of words) that match, the user clicks the button labelled "*". Pressing "0" completes the entry of the current word and creates a space, ready for the entry of the next word. The "C" button causes the last typed character to be cleared. A GUI class with these features is provided in the starter pack.

You can add other features if you wish, a drop-down menu for the list of words that match, up/down buttons to scroll through the menu etc. Feel free to practice working with the Swing API, as time permits.

Model-View separation

Thge interface between the GUI and the dictionary uses the *Observer* pattern to achieve a model-view separation. (This is also called the "Model View Controller" architecture, even though we don't necessarily use a separate "controller".) The dictionary, the text message etc. is to be encapsulated in a *Model* class. The GUI with the user interface components is built in a separate *View* class. The View class is an Observer (`implements Observer`) and the Model class is an Observable (`extends Observable`).

The Model

The Model class should be called `DictionaryModel`. It should implement the interface `DictionaryModelInterface`. It has two constructors:

```
public DictionaryModel(String dictionaryFile) throws java.io.IOException
public DictionaryModel() throws java.io.IOException
```

The first constructor takes the path name of a dictionary file and initialises the internal data structures. The second constructor uses a default dictionary file of your choice.

The `DictionaryModel` will be given the keys typed on the keypad via the `addCharacter` method. It should use the dictionary to predict the possible words (or prefixes of words) the user is trying to enter, and store them internally. The `nextMatch` method allows cycling through the matches to select the next matching word (or prefix) among the possible matches. In addition, the entire message being composed is also stored internally. The `acceptWord` method allows the current matched word to be accepted and added to the composed message.

Here are the required instance methods for the `DictionaryModel` class.

- `List<String> getMessage()`: Returns a list of the words typed in so far, including the last word (or prefix) which has not yet been accepted.
- `void addCharacter(char key)`: Adds a numeric key that has been typed in by the user. Extends the current word (or prefix) with possible matches for the new key.
- `void removeLastCharacter()`: Removes the last character typed in. This should remove the last character from the current match, but it would in general enlarge the possible matches for the current word.
- `void nextMatch()`: Cycles through the possible matches for the current word, i.e., changes the current word to the next match if there is one, or goes back to the first match otherwise.
- `void acceptWord()`: Accepts the current matched word and adds it to the composed message.

The View

A class called `View` is provided for you in the starter pack. It creates multiple copies of the GUI shown at the beginning of this handout, and links all of them to the same instance of the `DictionaryModel` that you build. Interacting with any one of them updates all the copies of the GUI in a consistent manner.

You are welcome to modify the `View` class and its subsidiary classes as you please.

Submission

You should test your `DictionaryModel` class using JUnit, as usual. Please submit a zip file containing your entire project, including the JUnit tests.

Hints

- Your GUI can be developed using any implementation of the `Dictionary` interface from Worksheet 3, even though the `TreeDictionary` class would work the best. It should not be necessary to modify any of your existing classes to complete this part.
- If you are not satisfied with your Dictionary classes, you can also use the `SampleDictionary` class that we provide on the Canvas page for this worksheet. This is a `jar` file (compiled binary). Instructions for incorporating it in your project will be posted on the Canvas page.
- You can use the demo application provided on the Canvas page to get a feel for how the user interface should work.
- You may want to implement more features such as number entry by keyboard, punctuation, editing already typed words and adding words to the dictionary, but make sure the basic functionality is working and committed to your submission first.
- You can find documentation for Swing on-line[1]

[1] A good place to start: http://java.sun.com/docs/books/tutorial/uiswing/