LINGUALINK

An Interactive Language Learning Tool

TABLE OF CONTENTS

PROJECT REPORT – (BEST DONE IN CONJUNCTION WITH DEV)

INTRODUCTION

The goal of the LinguaLink project is to develop a desktop application for language learners to diagram sentences in their target language and gain familiarity with grammatical rules. Specific requirements for this project include the ability for users to:

1. Add/remove words from the current project.
2. Move words from the word bank (inactive words) to the workspace (active words).
3. Move words around within the canvas.
4. Draw valid connections between words based on word part of speech.
5. Export an image of the current workspace diagram.
6. Save to and load from disk project configuration and state.

LinguaLink will allow language learners to get hands on, visual experience with their target language. It also provides them with immediate feedback. This serves to make the process of learning and practicing a languages grammar robust, interactive, and detailed.

LITERATURE REVIEW/BACKGROUND STUDY

This project required research into similar elements of each supported language. For English, the research process began with researching all parts of speech:

- Noun
- Pronoun
- Verb
- Adjective
- Adverb
- Preposition
- Conjunction
- Interjection

The decision was made to also support articles like “a” from diagram readability. If a sentence is a directed graph (which strongly aligns with the application architecture), then there must be some rules with which to determine valid edges in this graph. This required research on the semantics of the English language. For application robustness, it made more sense to enforce a potentially incomplete list of invalid constructions, rather than a massive list of valid constructions. The logic is that it is preferable that LinguaLink fail to flag unspecified invalid constructions than to fail to allow unspecified valid constructions, as this would result in an application with surprisingly (to the user) few possible sentences. That is, unless you manually enumerated all valid combinations (a painstakingly intense process for a project of this scope). The following are the initial invalid constructions implemented:

- Noun -> Adjective
- Adjective -> Verb

- Adverb -> Noun
- Preposition -> Verb
- Head of graph can't be a conjunction
- Article -> Article
- Preposition -> Preposition
- Conjunction -> Conjunction
- Pronoun -> Pronoun
- Noun -> Noun

Methodology

This project utilized certain principles of the Scrum framework. Essentially this boils down to a heavy emphasis on code first, test later, iteratively redesign and refactor. Given the scope and timeline of this project, it made sense to get moving sooner rather than later.

Implementation Details

This project was written in the language Java, and utilizes frameworks Swing for the graphical user interface and Junit and Mockito for unit testing and mocking to support those tests. The IntelliJ IDEA was utilized to organize and develop the project. Lastly Git was used for version control and GitHub was used for storage. This project is comprised of three different layers, each implemented on their own but dependent upon the previous layer.

The first layer is the base units of the application. These included low level abstractions necessary for use in higher layers of the project. These units include:

- Word – A class designed to store a word's value with its part of speech.
- WordBlock – A class to represent active words in the project's workspace. These contain a Word and a coordinate pair specifying their location on that workspace.
- PartOfSpeech – An enumeration containing English parts of speech valid within the context of this project.

Testing and Evaluation

The testing of this application relied on two main strategies.

The first is unit testing of each component in isolation to both catch deviations of actual implementation from desired implementation, as well as to increase confidence in later commits. This means that every component .java file should have a pair test file in the test directory of the project. For these unit tests, the project makes use of the Junit library. In addition to this, it uses Mockito to mock objects and function calls not crucial to the evaluation of the target component in isolation, as per the goal unit testing. Some major bugs discovered through this strategy:

The second testing strategy was to interact with the application via its user interface. This allowed the testing workflows to mirror the workflows or series of actions that might be taken by a user in a production context. Some major bugs found through this strategy:

- Window Sizing Bug – Running the project on a window showed that the initial sizing of the GUI window was too specific to work for different devices.

Results and Discussion

Conclusion

References and Appendices

User Manual – TODO (ONCE COMPLETE)

Installation

Features

Common Tasks

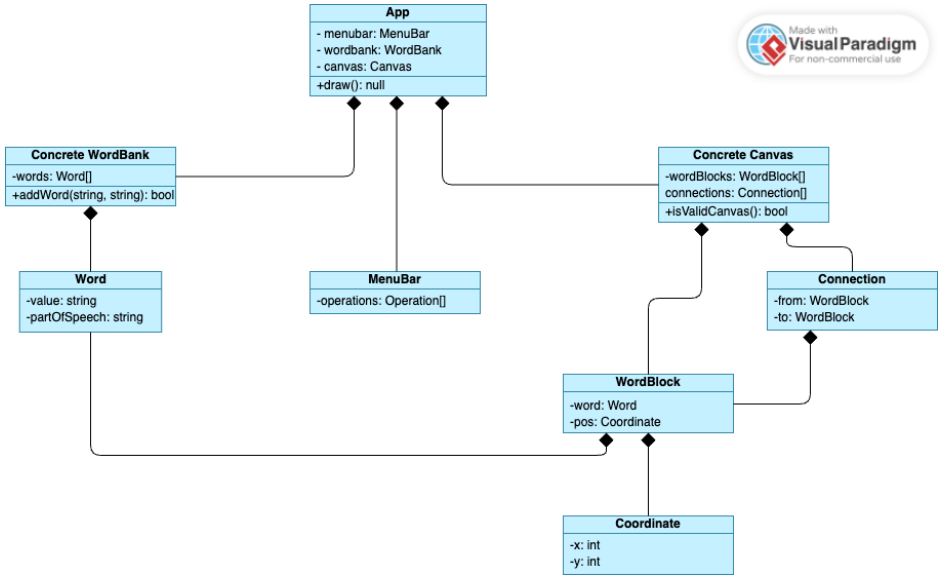
Troubleshooting

FAQs

Software Design Manual – TODO (PROBABLY BEST DONE AFTER COMPLETION)

Architecture Overview

See the class diagram below.



Design Patterns

Component Descriptions

Menu Bar: The menu bar resides at the top of the window. It is the users access point to high level controls. User stories include: “I would like to save my current project”, “I would like to export my current project as a PDF”, “I would like to export my current project as an image”, and “I would like to open an existing project saved to my hard drive”.

Word Bank: The word bank contains a list of words that are currently usage, but not yet placed on the canvas, within the current project. User stories include: “I would like to see what words are currently usable in my diagram”, “I would like to see what part of speech a word is”, and “I would like to add a new word for use in my diagram”.

Work Space: The work space contains current active diagrams. It also contains the language selection tool to switch to another language. Here users can place and order words, as well as draw links that form sentences. User stories include: “I would like to visually arrange certain words from my word bank”, “I would like to remove words from the work space”, “I would like to change my current project grammar to a French canvas”, and “I would like to draw a relationship between these two words”.

Diagrams

User Interface

The user interface design began with a simple wireframe of a desktop sized window. This window includes all thus far planned components.

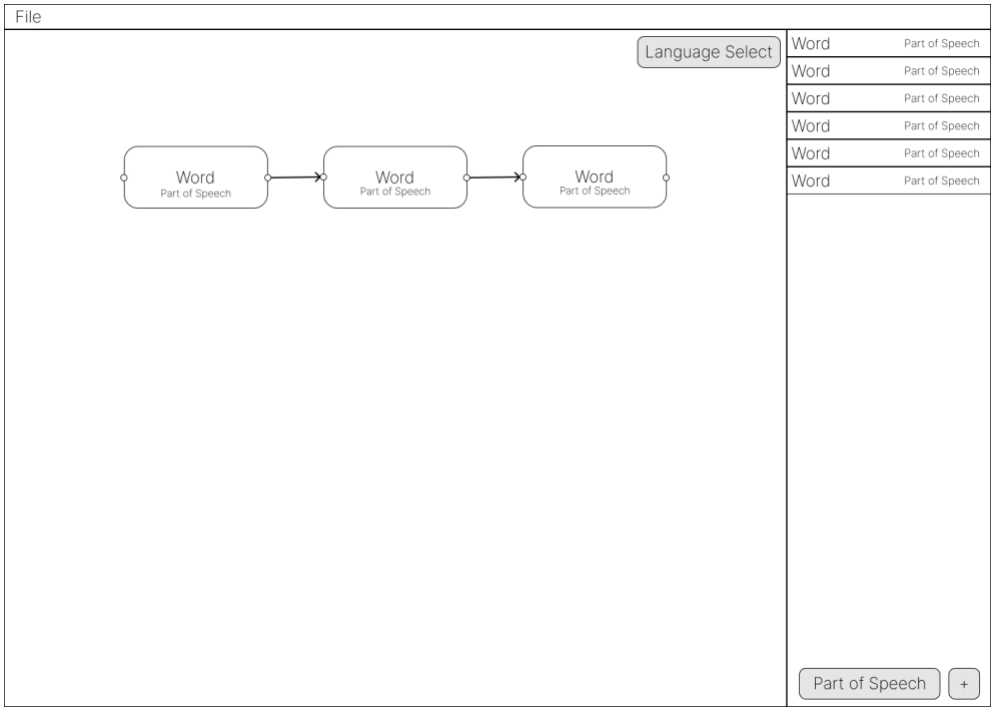


Figure 1

In the top right is the menu bar. The file dropdown menu will include the option to save the current file, open a new file, and export the current file as an image or PDF. The pane on the right-hand side is the word bank. This contains a list of all words present in the current workspace and a drop down to select a part of speech next to a button to add a new instance of that part of speech to the workspace. Users will be able to drag a word out of the word bank and place it on the canvas to the left. The canvas (the left pane) is where currently active words can be placed, arranged, and linked to form sentences. The arrows between words represent a transition between the previous word and the next. The application will notify users when the link is invalid (i.e. when a verb is connected, in sequence, to another verb).

Standard and Conventions

This project utilized the standards and conventions set for the in the Google style guide for Java. You can access this style guide on GitHub at https://google.github.io/styleguide/javaguide.html.

Appendices

ChatGPT Logs for Gradyn Nagle

See file: