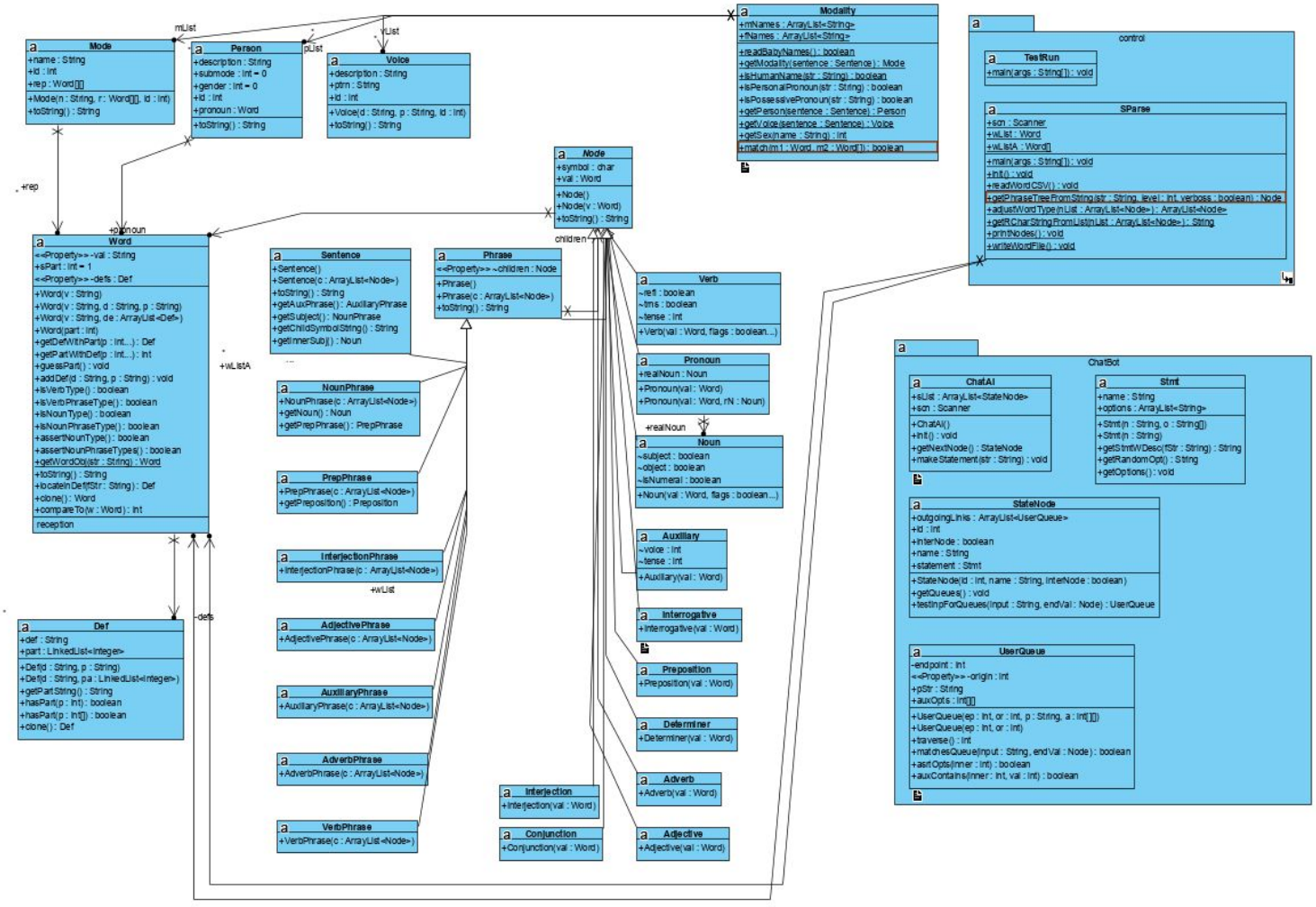
*Project Final - README*

# Chatbot

Team: Group 3 - Cody Tyerman, Adam Fox, Aleksander Konstantopulous, Matt Duroche

# *Class Organization:*

Below is our class hierarchy diagram in order to further explain the organization of each class:




**(See "*image1.png*" in GitHub repository for enlarged photo)**

# *Class Roles:*

**TestRun**: the entry point for the application, and the GUI control center (extends javafx application). Employs a singleton to allow control of JavaFX GUI elements from anywhere.

**ChatAI**: the class responsible for mimicking conversation

**StateNode**: The Node class for the graph of Chatbot behaviours

**UserQueue**: Class modelling links in the above-mentioned graph

**Stmt**: Statements that the bot can make

**SParse**: top level of speech parsing system which encapsulates and analyses the statements to determine metadata.
**Word**: wrapper class for dictionary word, the definitions associated with it, and the parts of speech associated with each definition
**Def**: struct to store a definition and part of speech qualifier
**Node**: Base class from which all other Sparse tree nodes extend
**Phrase**: extends node, represents a list of nodes which together form a type of phrase
About 15 subclasses of **Node** and **Phrase** representing the various parts of speech and phrases found in an English sentence. Little specific functionality is implemented in many cases, but room to do so exists.
**Sentence**: extends phrase, top-level class in the SParse phrase tree. It contains a noun phrase and a verb phrase typically. Can contain only a verb phrase, or only an interjection depending on the voice/mode
**Modality**: class which determines the modality, person and voice of the sentence/subject.
**Mode**: Enumerated instance of various types of mode (tense and optionality)
**Person**: same as above, but for 1st, 2nd, 3rd person, singular, plural, etc
**Voice**: imperative, declarative, interrogative, salutative (we made that last one up)
**InterludeConversation:** Handles the AIs response to user input during the "interlude conversations" which are triggered after the user responds a certain way to questions posed by the AI. During these interlude conversations the user is given a large degree of freedom as to what they can input, as the AI asks very open ended questions. The AI attempts to use NLP along with the SParse class in order to respond appropriately

## *Methods:*

### ***Sentence SParse.getPhraseTreeFromString(String):***

This method takes in a String of words (presumably forming a syntactically correct statement), splits it into its constituent words, encapsulates a matching word from the dictionary in a type of node matching it's likely role in a sentence, assigning each different type of word a different letter of the alphabet, allowing a regex match to be performed to determine which patterns of words forming phrases (themselves replacing words in other phrases higher up in the

hierarchy) can be encapsulated, and performing this operation (like a macro processor but in reverse) until the entire sentence is reduced to a single phrase (the Sentence type) containing all of the others. Further operations are performed on this end product in the Modality class to determine metadata.

***boolean UserQueue.matchesQueue(String, Node):***
This method is used to test the regex corresponding to each link on a node against the input given, in either the form of a String or a phrase tree as described in the method above.

***Mode Modality.getMode(Sentence):***
Checks a double array of nodes representing the possibilities for each of a number of predefined modes against the encapsulated phrase tree to determine which is employed in the sentence.

***int UserQueue.traverse():***
returns the identifier of the Node at the opposite end of the link.

## *Compilation Instructions:*

In order to compile our program, you will need to have JavaFX 11 along with the Java Development Kit 12.0.2. These can be found on their respective websites with the JDK being found at:
https://www.oracle.com/java/technologies/javase/jdk12-archive-downloads.html
and JavaFX 11 is found at:
https://gluonhq.com/products/javafx/
after these files have been downloaded and placed in the correct directories they should be added to the project build path in your Java IDE of choice.
The Stanford nlp libraries we used are:
-stanford-corenlp-3.9.2.jar
-stanford-corenlp-3.9.2.javadoc.jar
-stanford-corenlp-3.9.2.models.jar
-stanford-corenlp-3.9.2.sources.jar
-ejml-0.23.jar
These can be combined into a user library for ease of organization
After they have been added the program should compile and run without issue.
(A path mismatch was occurring in eclipse regarding the build path for

javafx.application and event, but that is just an IDE related issue, removing the import statement and adding it again resolves it)

## *List of Program Features:*

| Feature | How It Improved Response | Conversation Snippet |
|---|---|---|
| Interlude conversations | The addition of interlude conversations to the chatbots functionality allowed us to give the user more freedom with what they provided as input to the chatbot. It also served as a good opportunity to implement the Stanford NLP libraries such as NER and POS tagging . | Driver: Hi, my name is Mike, with EZ-cabs!<br>You: Hi MIke<br><br>Driver: Where to?<br>You: take me to the beach please<br><br>Driver: Is that somewhere out of town? |
| NLP | The interlude conversation class relies heavily on the NLP library to take things like places, people, professions etc… out of user input and analyze the sentiment of a sentence and use these things to respond more appropriately to inputs | Driver: Is that somewhere out of town?<br>You: yeah its just over the Brooklyn Bridge<br><br>Driver: the Brooklyn Bridge, ive been there before its beautiful! |
| SParse | The SParse class was our own implementation of a POS tagger. It allows the chatbot to identify modality and voice of a sentence to allow it to pick an appropriate response. | found match: start: 0, end: 1 with pattern#: 6 [go, to, {work}] :: ybf, i = 6, count = 1<br><br>found match: start: 1, end: 3 with pattern#: 6 [{go, to}, {work}] :: yy, i = 6, count = 2<br><br>voice : Imperative |