Technical Design Document - Team FEUD – Josh Ellington, Kevin Dustin, Stephen Fox, and Steve Ulibarri

**Executive Summary**

This program will be a very basic chat bot. Essentially it will simulate very basic Artificial Intelligence (AI) through interaction with the user. The program will accept input in the form of grammatically correct, basic English and respond appropriately—also in English.

The program will utilize dictionary files to detect words and determine the part of speech they belong to. (i.e. noun, verb, article, etc.) It will then attempt to construct a comprehensible sentence by analyzing how those parts of speech should interact based on a set of rules.

The program will then attempt to construct an appropriate response to the user’s input. It will use the words received from the user as well as words in its dictionary files to formulate a response in the form of a statement or question related to the user input.

The program will not be required to handle incorrect grammar, slang, profanity, obscure vocabulary or non-English words. Once the minimum requirements are met adding that functionality is optional.

**Definitions**

Sentence: a grammatically correct English sentence

Word: a valid English word

Part of Speech: any individual valid single piece of the English language including words and punctuation

Punctuation: any valid English punctuation mark

Noun: any valid English noun (person, place, thing or idea)

Verb: any valid English verb (action word)

Helping Verb: a verb used in conjunction with another verb (i.e. to indicate tense)

Article: any valid English article (used before nouns)

Noun Phrase: a group of words constituting a single ‘noun concept’ (i.e. article + noun)

Verb Phrase: a group of words constituting a single ‘verb concept’ (i.e. main verb + any helping

**Requirements Map**

|  |  |
| --- | --- |
| Requirement 1 | Must be able to detect basic parts of speech |

This requirement is handled through our use of dictionary files. Early on in our development we acquired a massive list of English words, arranged alphabetically and tagged with a character indicating any parts of speech that any one word could be. We wrote a parser and sorted this large file into separate files containing nouns, verbs, prepositions, and articles. In our WordProcessor class the user’s input is split into individual words. Each word is sent to our WordTagger class which attempts to find it in these files. If found, the program will wrap the word in the appropriate PartOfSpeech object. If the word is not found the program will indicate that this is the case and the word can be manually added to the appropriate file.

|  |  |
| --- | --- |
| Requirement 2 | Must be able recognize a valid sentence as defined by our grammar rules |

This requirement is handled by the languageProcessor class which uses a basic natural language processing algorithm. According to our grammar rules a valid sentence is composed of a noun phrase and a verb phrase. Noun phrases and verb phrases are composed of various, grammatically correct, combinations of nouns, articles, verbs, and prepositions. The algorithm is able to discern proper grammar by knowing when and how to perform these combinations.

|  |  |
| --- | --- |
| Requirement 3 | Must be able to interact through the console |

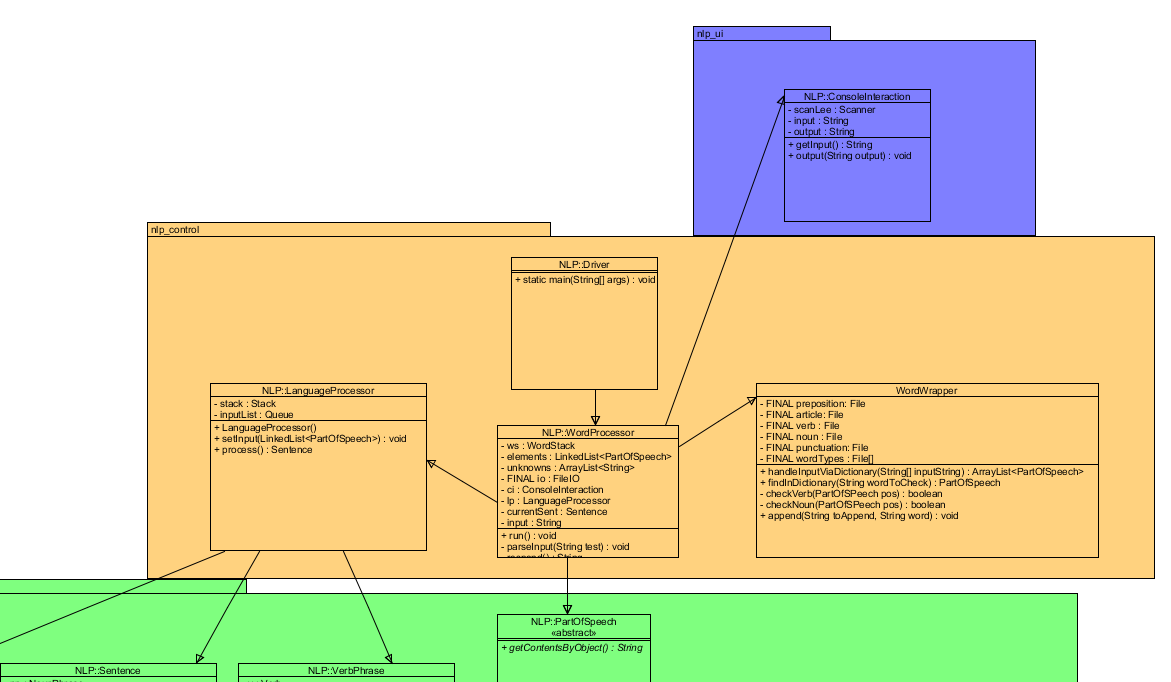
This requirement is met by our ConsoleInteraction class. This class is able to obtain input from the user and print output statements to the user.

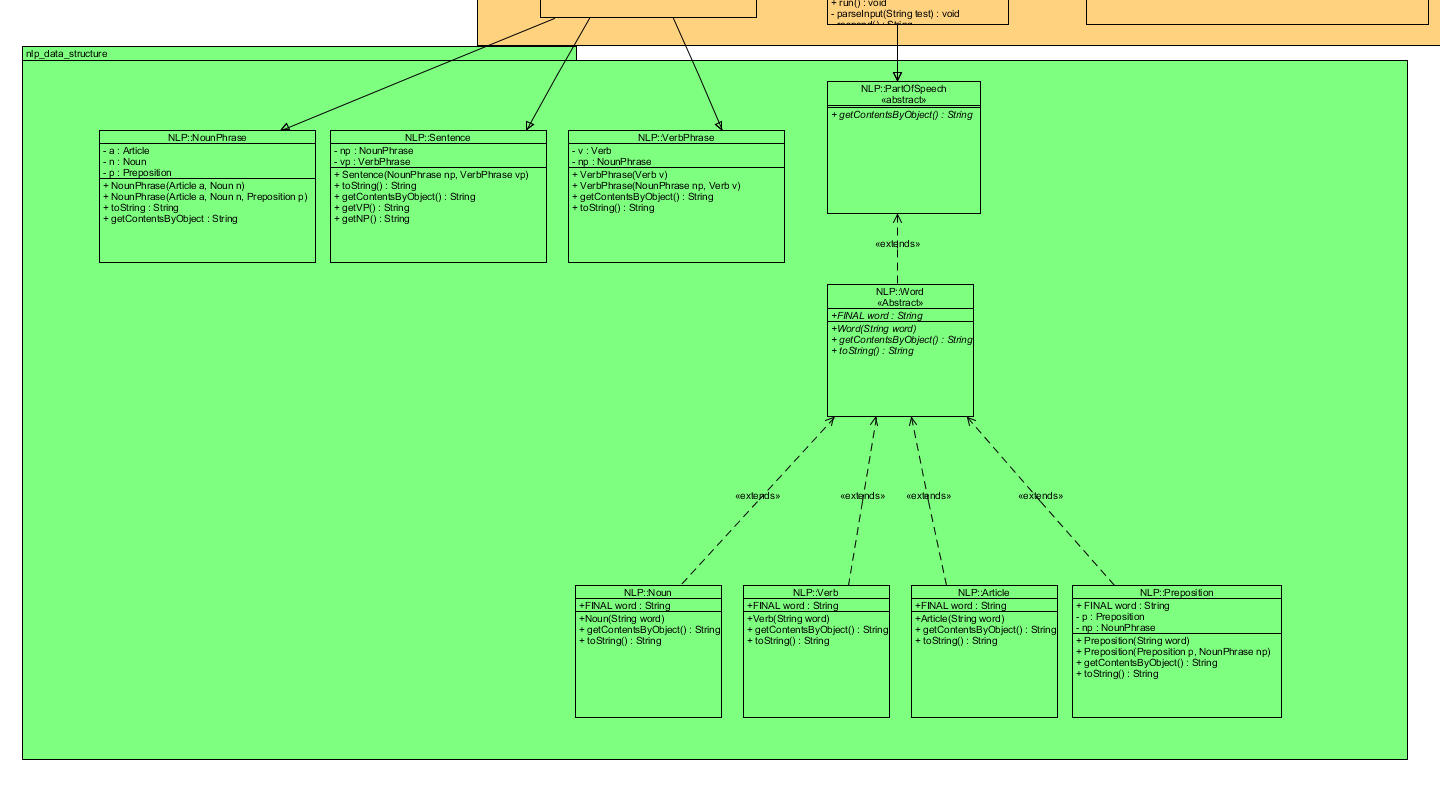
|  |  |
| --- | --- |
| Requirement 4 | Must Respond with relevant statements or questions |

This requirement is handled in our WordProcessor class. This class contains methods that are able to construct replies out of valid sentences. These replies are built out of a selection of pre-built response “templates”. The methods are able to grab relevant parts of the valid Sentence object and place them into the templates which are then printed to the user. The methods will also attempt to conjugate and or pluralize nouns and verbs in order to produce grammatically correct replies. However, this is not always successful due to the complexity of English verb conjugation. On the bright side the output is often hilarious.

|  |  |
| --- | --- |
| Requirement 5 | Must be able to tell the user when it is unable to understand the input |

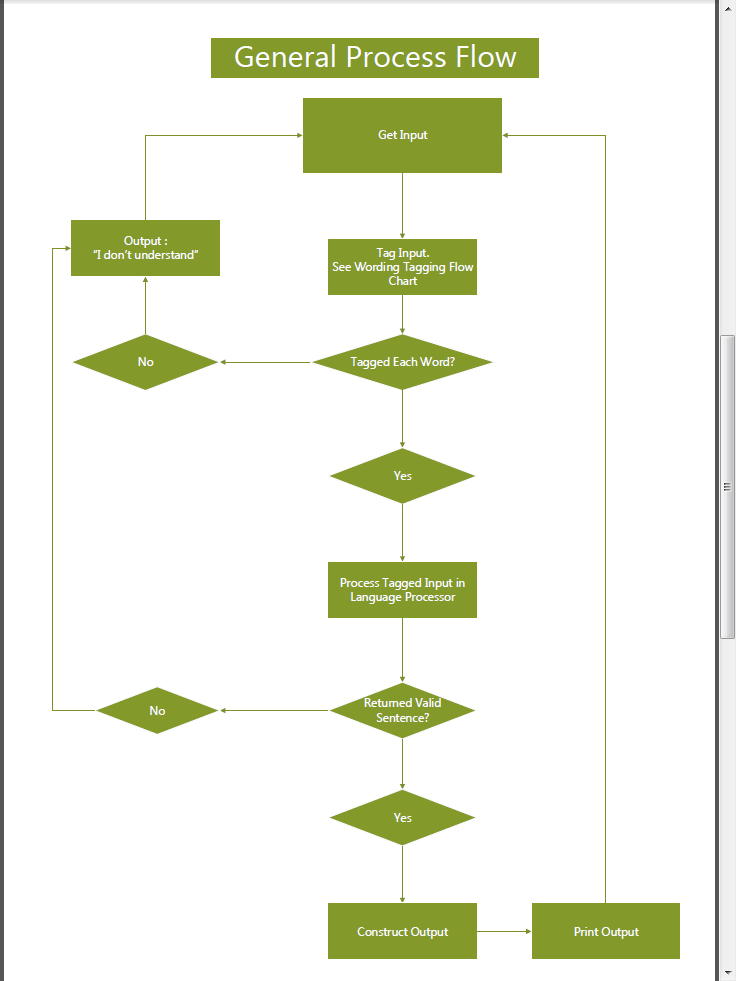
This requirement is handled in two places. First, in the WordTagger class, if the program is unable to find an input word in any of the dictionary files it will alert the user that it was unable to understand as it technically doesn’t “know” the word. Second, in the event that each word in the users input is tagged appropriately but the LanguageProcessor class is unable to combine these words into a valid sentence the LanguageProcessor class will return null. When the word processor receives detects this null it will then prompt the user that it was unable to understand the input.

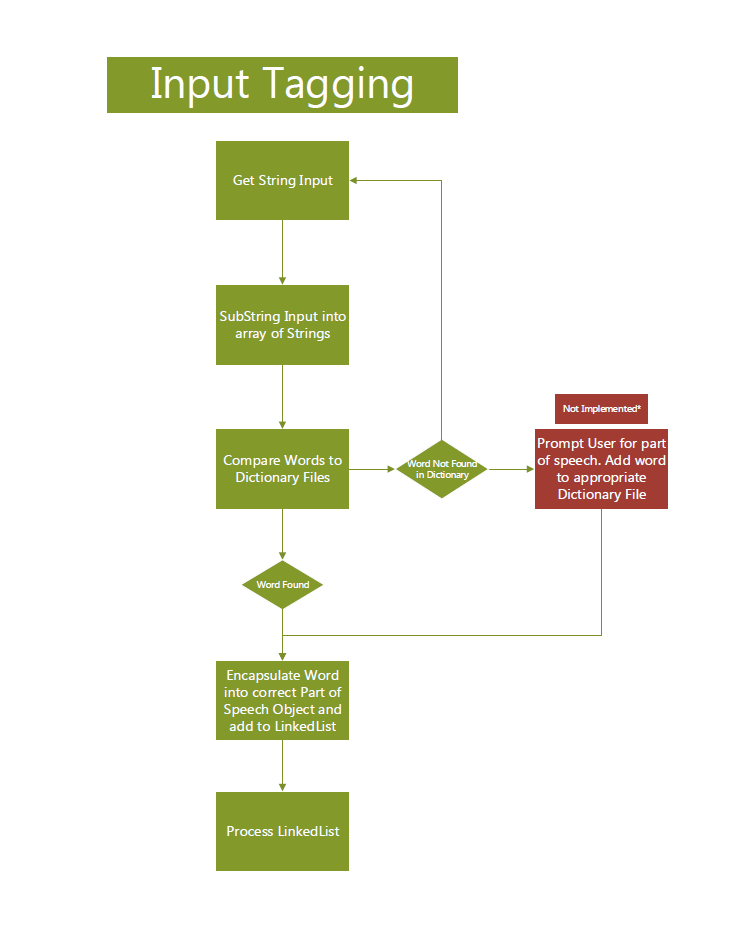
**UML**

**The above picture is the control (orange) and user interface (blue) packages.**

**This is our data structure in green.**

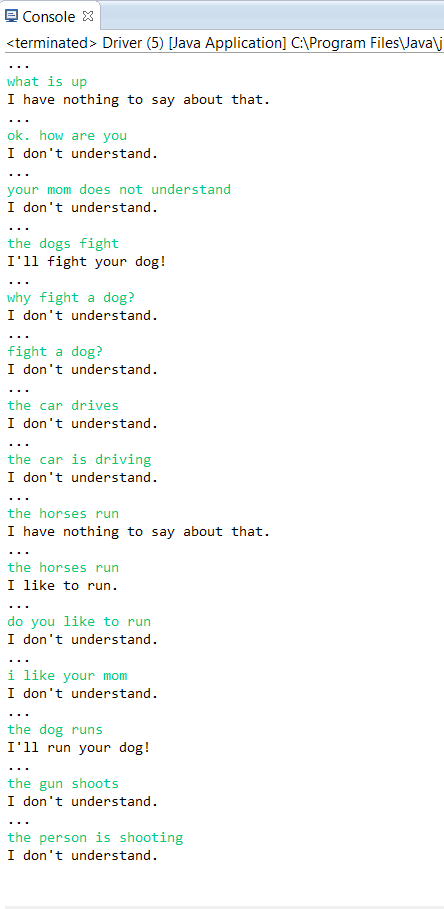
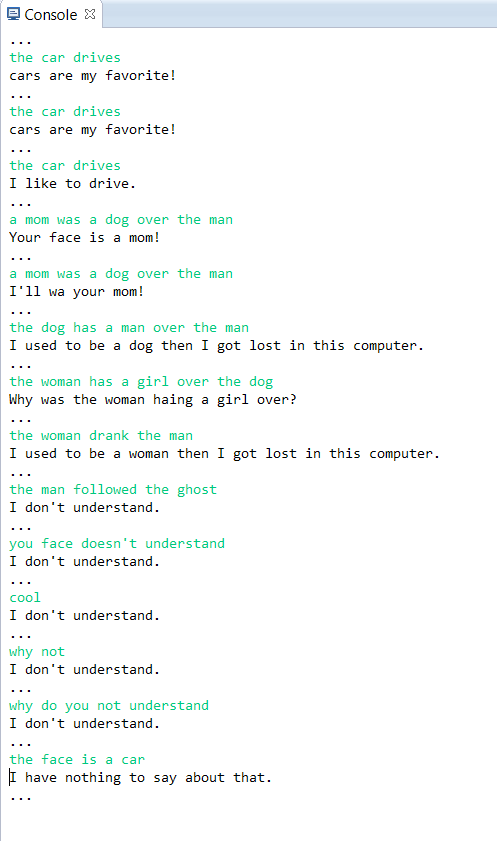
**Process Flow Diagram**

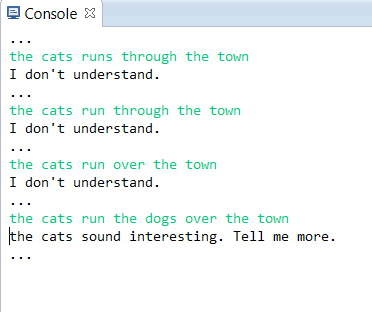
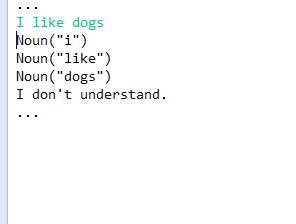
****

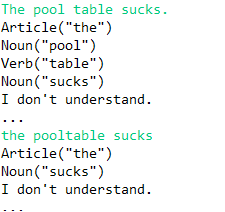
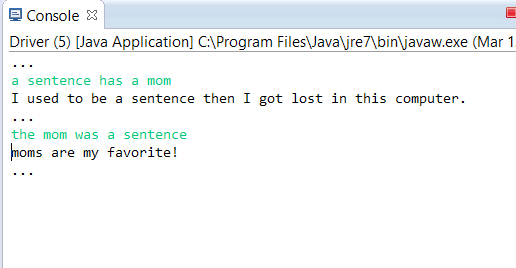
****

**Database**

****

**UI Screenshots**

****

****

These are all screenshots of “conversations” with our chat bot demonstrating what it can and can’t understand.

**Detailed Test Cases**

Test 1:

Precondition: Program is accepting user input (weather on startup or mid conversation).

1: The test class sends the String, “The cats chase the dogs.” to the controller to test.

2: The WordProcessor splits the word into individual words, removing punctuation as found.

3: Each word is compared to our dictionary files in the WordWrapper. It is then created as a verb, noun, article, or preposition object.

4: The LanguageProcessor then attempts to create a sentence out of the wrapped up words. If it can form a proper sentence, that sentence is returned. Else, null is returned.

5: The WordProcessor than takes either the verb or noun and responds with a sentence referring to that word.

Result: The program prints out these possible responses:  
 “Why were the cats chasing the dogs?”

“Your face is a cat!”

“Cats are my favorite.”

“I used to be a cat, then I got lost in this computer.”

“I like to chase.”

“I’ll chase your cat.”

“The cats sound interesting, tell me more.”

Test 2:

Precondition: Program is accepting user input (weather on startup or mid conversation).

1: The test class sends the String, “the fish chased the cat over the bridge.” to the controller to test.

2: The WordProcessor splits the word into individual words, removing punctuation as found.

3: Each word is compared to our dictionary files in the WordWrapper. It is then created as a verb, noun, article, or preposition object.

4: The LanguageProcessor then attempts to create a sentence out of the wrapped up words. If it can form a proper sentence, that sentence is returned. Else, null is returned.

5: The WordProcessor than takes either the verb or noun and responds with a sentence referring to that word.

Result: The program prints out these possible responses:  
 “I’ll chas your fish!”

“the fish sounds interesting, tell me more.”

“I like to chas.”

“Your face is a fish!”

“fishes are my favorite.”

“I used to be a fish, then I got lost in this computer.”

“Why was the fish chasing the cat over?”

Test 3:

Precondition: Program is accepting user input (weather on startup or mid conversation).

1: The test class sends the String, “Teh cats chase the dogs.” to the controller to test.

2: The WordProcessor splits the word into individual words, removing punctuation as found.

3: Each word is compared to our dictionary files in the WordWrapper. It is then created as a verb, noun, article, or preposition object.

4: The LanguageProcessor then attempts to create a sentence out of the wrapped up words. If it can form a proper sentence, that sentence is returned. Else, null is returned.

5: The WordProcessor than takes either the verb or noun and responds with a sentence referring to that word.

Result: The program prints out these possible responses:  
 “I don’t understand.”

Our program doesn’t understand typos.

Test 4:

Precondition: Program is accepting user input (weather on startup or mid conversation).

1: The test class sends the String, “Cats are running.” to the controller to test.

2: The WordProcessor splits the word into individual words, removing punctuation as found.

3: Each word is compared to our dictionary files in the WordWrapper. It is then created as a verb, noun, article, or preposition object.

4: The LanguageProcessor then attempts to create a sentence out of the wrapped up words. If it can form a proper sentence, that sentence is returned. Else, null is returned.

5: The WordProcessor than takes either the verb or noun and responds with a sentence referring to that word.

Result: The program prints out these possible responses:  
 “I don’t understand.”

Our program requires an article to precede a noun.

Test 5:

Precondition: Program is accepting user input (weather on startup or mid conversation).

1: The test class sends the String, “The cats are running.” to the controller to test.

2: The WordProcessor splits the word into individual words, removing punctuation as found.

3: Each word is compared to our dictionary files in the WordWrapper. It is then created as a verb, noun, article, or preposition object.

4: The LanguageProcessor then attempts to create a sentence out of the wrapped up words. If it can form a proper sentence, that sentence is returned. Else, null is returned.

5: The WordProcessor than takes either the verb or noun and responds with a sentence referring to that word.

Result: The program prints out these possible responses:  
 “I don’t understand.”

It will not understand two verbs back to back.

Test 6:

Precondition: Program is accepting user input (weather on startup or mid conversation).

1: The test class sends the String, “I like dogs.” to the controller to test.

2: The WordProcessor splits the word into individual words, removing punctuation as found.

3: Each word is compared to our dictionary files in the WordWrapper. It is then created as a verb, noun, article, or preposition object.

4: The LanguageProcessor then attempts to create a sentence out of the wrapped up words. If it can form a proper sentence, that sentence is returned. Else, null is returned.

5: The WordProcessor than takes either the verb or noun and responds with a sentence referring to that word.

Result: The program prints out these possible responses:  
 “I don’t understand.”

It doesn’t handle pronouns.

Test 7:

Precondition: Program is accepting user input (weather on startup or mid conversation).

1: The test class sends the String, “The fish chase the dogs over the bridge through the town.” to the controller to test.

2: The WordProcessor splits the word into individual words, removing punctuation as found.

3: Each word is compared to our dictionary files in the WordWrapper. It is then created as a verb, noun, article, or preposition object.

4: The LanguageProcessor then attempts to create a sentence out of the wrapped up words. If it can form a proper sentence, that sentence is returned. Else, null is returned.

5: The WordProcessor than takes either the verb or noun and responds with a sentence referring to that word.

Result: The program prints out these possible responses:  
 “The fish sound interesting. Tell me more.”

“Your face is a fish!”

“I like to chase.”

“I used to be a fish, then I got lost in this computer.”

“Why was the fish chasing the dogs over?”

“fishs are my favorite!”

“I’ll chase your fish!”

Test 8:

Precondition: Program is accepting user input (weather on startup or mid conversation).

1: The test class sends the String, “The, cats, chase; the$ dogs.” to the controller to test.

2: The WordProcessor splits the word into individual words, removing punctuation as found.

3: Each word is compared to our dictionary files in the WordWrapper. It is then created as a verb, noun, article, or preposition object.

4: The LanguageProcessor then attempts to create a sentence out of the wrapped up words. If it can form a proper sentence, that sentence is returned. Else, null is returned.

5: The WordProcessor than takes either the verb or noun and responds with a sentence referring to that word.

Result: The program prints out these possible responses:  
 “Why were the cats chasing the dogs?”

“Your face is a cat!”

“Cats are my favorite.”

“I used to be a cat, then I got lost in this computer.”

“I like to chase.”

“I’ll chase your cat.”

“The cats sound interesting, tell me more.”