# Ozi Yesufu Assignment 3

# (additional features at the bottom)

# README

## Overview

ACTUALLY CHAT is a chat bot that is able to carry a conversation with a real person for at least 30 turns, a turn being  a statement or question by either the bot, or the person, in a set of specific topics. The program consists of a database of keywords it uses to match the sentence it needs to reply to and provide an appropriate response.

## Scope

The bot is able to simulate the conversation of a first date.

## Design

### Database

Table: keywords

| Column | Type | Example Data | Notes |
|---|---|---|---|
| id | INT( 10 ) UNSIGNED auto_increment PRIMARY KEY | 1 | An id to reference from the responses table |
| keywords | VARCHAR | Hi\|hello | Lowercase, equivalent keywords separated with a "\|"<br>Keywords can be a phrase like "have you" or "will you ever" for greater flexibility |
| type | enum( 'word', 'phrase' ) | word | Can look for a word or a whole phrase for greater flexibility and matching.<br>Whole phrases should appear higher in the list than individual words |

| | | | |
|---|---|---|---|
| sentence_location | Enum( ‘starts-with’, ‘ends-with’, ‘contains’ , ‘exact’) | ends-with | Where in the sentence should the word or phrase occur |
| word_location | Enum(‘starts-with’, ‘ends-with’, ‘contains’, ‘exact’) | Contains | Where in the individual word of the sentence should the keyword occur. Only use this flag if the type is ‘word’. If Phrase is the type, then we don’t’ need to match on the word location.<br>If word is the type, then we split the sentence into words and check against each word.<br>Example: if ‘feel’ is the keyword and ‘starts-with’ is the condition then if the sentence has ‘feeling’ as a word, it will match, but if the condition was ‘exact’ then it wouldn’t match. This allows to match on very specific words, or do a more general match against word fragments like ‘psycho’ and ‘starts-with’ just matches any word that starts with psycho and might have a general canned response. Ie. We don’t care exactly what word it is. |
| Weight | INT | 29 | Higher numbers appear earlier in the list so have more of a chance of getting matched. The first match will be the row that’s used. Separate each weight by multiple of 10, to allow for easily adding up to 9 additional words inbetween existing words for easy flexibility. Ideally we would have some interface that does the weighting for us using drag and drop to order the keywords. |

Table: responses

| | | | |
|---|---|---|---|
| ref_id | INT( 10 ) UNSIGNED<br><br>FOREIGN KEY references keywords( id ) | 1 | Links the response to a keyword in the keywords table. There could be many different responses per keyword |
| keywords | VARCHAR | doing\|mooing | Individual keywords separated by a “\|” |

| | | | |
|---|---|---|---|
| question | Enum( 'yes', 'no', 'no-preference' ) | Yes | Indicates whether the sentence entered by the user should be a question, or not , or whether it matters at all. The program will continue to go down the list of responses until it finds one that matches all the conditions and will use that one. |
| Response | VARCHAR | That's cool.\|YOLO | Randomized responses separated by "\|" |
| Weight | INT | 400 | Same as the keywords table. Higher weight keywords will be evaluated first |

## Data File

The database will be used as a tool to create a data file to ship with the program. The data file will be an exact representation of the database and be used by the java program to build the knowledge bank to store in memory.

Syntax:

The entire information for a keyword should be stored on one line so the java can easily split individual keyword information on "\n" or "\n\r"

<keyword-string>#<type>#<sentence-location>#<word-location>#<weight>#<responses>

Syntax for <responses>

[R]<response-keyword-string>\\<response-question-flag>\\<responses-string>\\20

For example, a full keyword line (Note, this is all still on one line) with two possible responses might look like this:

hi|hello#word#contains#starts-with#520#[R]friend\\no\\hi friend!|hello there[R]\\no\\go away\\40

The java program can then split on '#' to get an array where

Array[ 0 ] = the keywords

*Then split on "|" to store individual words or phrases in a list or array*

Array[ 1 ] = the type

Array[ 2 ] = the sentence location

Array[ 3 ] = the word location

Array [ 4 ] = the weight

Array [ 5 ] = The responses

Array[ 5 ] then needs to be split on “\\” to get a list of responses ->

Responses[ 0 ] = the response keywords

*Needs to be further split on “|” to store the individual words in a list or array.*

Responses[ 1 ] = the question flag
Responses[ 2 ] = The responses string

*Needs to be further split on “|” to store the response options in a list or array.*

*These response options should be similar and will be randomized to make the bot a little more human-like. Ie. A human is not going to answer the same question in the same way every time, necessarily.*

Responses[ 3 ] = The weight of the response.

## JAVA CLASS STRUCTURE

- package actuallychat
    - package actuallychat.main.java
        - package actuallychat.main.java.chat
            - class main
            - interface Responder
            - interface Chat
            - class AbstractKeyword
            - class AbstractResponse
            - class ActuallyResponder
            - class ActuallyChat
            - class ActuallyKeyword
            - class ActuallyResponse
    - package actuallychat.main.test
        - Unit Tests

## Class Description

class main

- Runs the program. Makes a call  for user input or starts the chat, etc.

Interface Responder

- void respond( String input );
- void greet();

Interface Chat

- void chat();
- String getUserInput();

Class AbstractKeyword

- static final enum LOCATION { STARTS_WITH, ENDS_WITH, CONTAINS, EXACT };
- String[] keywords;
- String type;
- String sentenceLocation; // uses LOCATION enum;
- String wordLocation; // uses LOCATION enum;
- Int weight;
- List<AbstractResponse> responses;// sorted by weight

Class AbstractResponse

- String[] keywords;
- String questionFlag;
- String[] responses;
- Int weight;

Class ActuallyResponder implements Responder

- List<AbstractKeyword> keywords; // sorted  by weight

Class ActuallyChat implements Chat

Class ActuallyKeyword extends AbstractKeyword

Class ActuallyResponse extends AbstractResponse

## THE PROGRAM PROCESS

Parse the text input file with all the keywords and store as a list of Keywords in the responder.

Take user input, and call Responder.respond( sentence );

Parse the sentence to determine if it is a question and store as a Boolean flag or enum.

Store the sentence as a string and as an array.

Loop through the list of keywords until one matches. If none match, use a default response.

For each keyword object, check if either of its key words or phrases match anything in the sentence. If ‘word’ is the type, loop through the list of words in the sentence and see if any match based on the sentence_location and word_location flags. If they match, the proceed to Choose a Response. If they don’t match, go to the next keyword.

### CHOOSE A RESPONSE.

Once a top-level keyword or phrase is matched, look at the list of responses. Go down the list, and check the user sentence for a keyword specified in the response, and check the question flag of the response. Does the response care whether or not the sentence is a question? If the condition matches, then choose a random response from the response array. Output this to the user and wait for additional input from the user.

## Assignment 3 additional Features

This project is an addition to assignment 2. The base project was not changed.

3 new programming features were added.

1. Named entity recognition

-since it is a first date simulator, people some times will choose to talk about other people as a topic on that date. At that point people will know the common names of famous people. I added in Named entity recognition for when the user inputs something about a well known person. The system will recognize this well known person's name and return the appropriate response.

ex.

```
Welcome to Chat.
hello
Hello.
I know Tim Horton
I also know Tim Horton
```

2. 5 responses outside the two topics

-For a more natural conversation, the system now has 5 new responses for when the under has input something outside of the topics.

ex.

```
What's your birthday?
My birthday is February 29th, 1983. I'm getting old...
you're annoying
That's not very nice. I don't like you
What's your birthday?
My birthday is none of your business.
```

3. Extra topic

-As an extra topic I added the situation where the first date has gone wrong. For example, when the user says something rude the not switches mood to being angry at the user. All responses to the user now have an angrier and more rude tone. This will continue until the user says sorry. At this point it changes back into a better mood.

ex.

```
huh
What does that mean?
huh
Aren't we on a date? Lets talk about that.
huh
How about we focus on us
huh
Aren't we on a date? Lets talk about that.
```