CS1006 Project 1

Introduction

This project was about creating an Eliza style program. It is intended to simulate a conversation with a psychiatrist. Eliza itself has no understand of English, but is given a set of keywords to search input strings for and thus generate a response. In short, Eliza is all about examining and handling of text. Eliza consists of two parts: A script and an engine.

Glossary

Term	Meaning	Example
Pre-substitution rule	Substitutions applied before matching input sentences with keywords. It standardizes the user input. Its purpose is to make there have to be less keywords and decomposition rules, because substitutions are made before scanning.	"I'm" becomes "I am" "also" becomes nothing.
Keyword	Keywords are looked for in input sentences. If an input sentence contains a keyword the engine moves on to matching the sentence with a decomposition rule for that keyword. Keywords have a set priority so that if two keywords are found in an input string, the engine knows which one to choose.	"think" "am"
Decomposition rule	Describes how to break down user input so that so that the engine can choose a reasonably response (i.e. reassembly rule). Stars can be used to represent a chunk of text input by the user in a decomposition rule. This star can be referenced in the reassembly rule.	"I think *" "I am *"
Reassembly rule	Describes how to put together a reply, sometimes using parts of the user's input, represented in the rules by a placeholder character (in our case numbers 1 and 2) which references a star in the decomposition rule.	"Why do you think 1?" "What makes you 1?"
Post-substitution rule	Substitutions applied once output is being generated after a reassembly has been chosen and a reference to the decomposition rule was made. Its purpose is to fix the semantics of an output message during reassembly.	"I'm" becomes "I am" "I" becomes "you"

Other responses	Other responses are stock phrases which Eliza uses when no keywords are found in input sentences to advance the conversation.	"Please tell me more." "Can you elaborate on that?"
Welcome message	A simply greeting from Eliza. Ideally asking the user a question to start of the conversation.	"Hi, I'm Eliza. What do you want to talk about?"
Quit commands	Words or phrases looked for in inputs that trigger a final message and the termination of the program.	"bye" "goodbye"
Final messages	The last thing the users says to the user before it terminates.	"Goodbye. Have a nice day."
Synonym keywords	Keywords that have the same or similar decomposition and reassembly rules.	"mother", "father", "mum", "dad".
Decomposition-rule-synonyms	Decomposition rules that have the same reassembly rules	"My problem is *", "My problem has to do with *"

How it works: The Basics

The script contains a number of pre- and post-substitution rules, keywords, and various other responses. The engine reads and interprets the script and takes in input sentences and generates responses based on the scripts content. It first applies pre-substitution rules, then scans for keywords, matches the sentence with a decomposition rule and then uses a reassembly rule associated with that decomposition rule. It then applies post-substitution to any copied text from the input and finally outputs a response to the user's input. If no keyword is found, it responds with a stock phrase such as "Please tell me more."

Sample conversation

This sample conversation illustrates many of the features of the Eliza program.

```
Eliza: Hello, I'm Eliza. What would you like to discuss?
>>Hi Eliza. I want to talk about my problem.
Eliza: Tell me more about your problem.
>>My problem is my parents getting divorced.
Eliza: Please tell me more about your parents getting divorced.
>>I think my father cheated on my mother.
Eliza: Are you close to your father?
>>Yes, but I am suspicious of him.
Eliza: Ok.
>>I don't trust him.
Eliza: I need you to elaborate on that.
>>I'm also certain my mother stole money from him.
Eliza: Tell me more about your mother.
>>She's a good person, but sometimes dishonest.
Eliza: Please tell me moer.
>>She lied about where she got the money.
Eliza: You need to elaborate on that.
>>She said she won it in the lottery.
Eliza: Can you elaborate on that?
>>I'd give good money to see her get what she deserves.
Eliza: Why would you give good money to see her get what she deserves?
>>Because I believe thieves should be punished.
Eliza: Have you always believed thieves should be punished?
```

```
>>Yes, I hate all criminals.
Eliza: Why do you hate all criminals?
>>I think people should follow the law.
Eliza: Tell me more about people.
>>All people are different.
Eliza: Are there any people close to you?
>>I am very close to my brother.
Eliza: Why are you very close to your brother?
>>Because he looks up to me.
Eliza: I need you to elaborate on that.
>>I believe I am his idol.
Eliza: Have you always believed you are his idol?
>>Sorry, I have to go now. Goodbye.
Eliza: Goodbye, have a nice day.
```

Explanation of Sample Conversation

```
Eliza: Hello, I'm Eliza. What would you like to discuss?
```

Eliza starts the conversation and asks a question to get it going.

```
>>Hi Eliza. I want to talk about my problem.
Eliza: Tell me more about your problem.
```

Eliza matches the input sentence with a keyword/decomposition rule and uses the reassembly rules to generate a response asking for more information.

```
>>My problem is my parents getting divorced.
Eliza: Please tell me more about your parents getting divorced.
>>I think my father cheated on my mother.
Eliza: Are you close to your father?
```

It continues the conversation and detects other keywords and moves the conversation in a different direction.

```
>>Yes, but I am suspicious of him.
Eliza: Ok.
```

Since it does not locate any keywords in the longer sentence-part it uses the reassembly rule for "yes" and generates a very basic response.

```
>>I don't trust him.
Eliza: I need you to elaborate on that.
```

Eliza does not find any matching keywords and therefore uses one of the other-responses to further the conversation.

```
>>She's a good person, but sometimes dishonest.
Eliza: Please tell me moer.
```

Again, Eliza detects no keywords and uses an other-response. A typo generator makes the word "more" into "moer", by swapping two adjacent characters. This makes Eliza seem more human-like.

```
>>I'd give good money to see her get what she deserves.
Eliza: Why would you give good money to see her get what she deserves?
```

In this case it uses one of the pre-substitution rules to make "I'd" into "I would" because scanning for keywords. It then finds the keyword "would" and generates an appropriate response.

```
>>I believe I am his idol.
Eliza: Have you always believed you are his idol?
```

Here, while Eliza is applying the reassembly rule, it uses one the post-substitution rules to change "I am" into "you are".

```
>>Sorry, I have to go now. Goodbye.
Eliza: Goodbye, have a nice day.
```

The user inputs a quit-command which is used to trigger a final message and terminate the conversation.

Design

When designing the program, consideration has to be given to two main aspects: The syntax of the script and the implementation of the engine.

Script

The syntax of the script is important to think about flexibility and readability. Flexibility meaning that it can have several keywords with the same decomposition rules and reassembly rules, and readability meaning how easy it is to interpret the script by a person. When making these decisions focus was given to the readability rather than the efficiency of the parser. The reason for this is that whilst a more complicated parsing method had to be implemented, it made the editing of the script much easier. Since more time would be spent developing the script rather than this one method this was viewed as a reasonable trade-off. Therefore it was decided to represent corresponding keywords, decomposition and reassembly rules on separate lines.

Below is an explanation of each section of the script file.

Welcome messages

These are final/greeting messages and each one is on a separate lines.

```
;Welcome
Hello, I'm Eliza. What would you like to discuss?
```

Final

These are final/goodbye messages. They are all on separate lines.

```
;Final
Goodbye, have a nice day.
```

Pre-substitution rules

These are pre-substitution rules. They are separated by tabs.

```
;Pre
I'm    I am
```

Post-substitution rules

These are post-substitution rules. They are separated by tabs.

```
;Post
I am    you are
```

Keywords

This line marks the beginning of the keyword section of the script file:

```
;Keywords
```

Keywords are in lines starting with “k:” in the keyword section.

```
k:believe 3
```

The first word (believe in this case) is the keyword and the number (3) represents the priority of it. The priority can range between 1 and 10, 1 being the highest and 10 being the lowest priority.

```
d:I believe *
```

"d:" marks the beginning of a line containing a decomposition rule. Each decomposition rule can have several reassembly rules which are on separate lines, each starting with "r:" and directly following the line containing the decomposition rule:

```
r:Have you always believed 1?
r:Are your beliefs important to you?
r:Beliefs are very important. What else do you believe in?
```

Keyword synonyms are separated by spaces in the keyword definition line, and, as always, followed by the priority (2):

```
k:mother father mum dad mom daddy 2
d:My $ is *
r:Why would you say your $ is 1?
r:It is interesting that you say your $ is 1.
r:Please tell me more about why your $ is 1.
```

In the first line in the script fragment above, keywords-synonyms are defined. In the decomposition/reassembly rules the keywords-synonyms are referenced with dollar-signs ($). So if the input sentence is for example "My father is very kind." It could be reassemblied as "Why would you say your father is very kind?"

A keyword can have several decomposition rules, and can also have synonym-decomposition rules, which have the same reassembly rules.

```
k:sorry 4
d:I am sorry because *
r:You say you are sorry because 1?
r:You don't need to apologise
d:He is sorry */He is sorry
r:How does his apology make you feel?
r:What do you think of him now?
r:Do you think his apology should be accepted?
```

As seen in the line "d:He is sorry */He is sorry", decomposition-synonyms are separated by slahes.

Other responses

These are other responses. They are represented on separate lines.

```
;Other
Can you elaborate on that?
```

Quit commands

The quit commands are defined in the section of the script file beginning with:

```
;Quit
```

Design: Engine

When designing the engine consideration of data representation, file parsing, text manipulation and sorting algorithms has to be given.

Class	Purpose	Notable Design Decision(s)
eliza.Engine	Contains the Eliza engine. Is used to generate responses to input strings using the script.	- The Keywords are stored in an ArrayList.
eliza.Runner	The main class and therefore contains the only main method. Used for running the program.	N/A?
eliza.exception.FileFormatException	An exception marking a syntax error in the script file. It can be thrown when parsing the script file.	Subclass of Exception.
eliza.methods.RandomMethods	A class containing the random number generator methods used by some of the other classes, most notably eliza.Engine.	
eliza.wrapper.Decomposition	Wrapper for the decomposition rules and associated reassembly rules.	Reassembly rules are stored in a StringArrayRandomizer.
eliza.wrapper.Keyword	Wrapper for keywords, their priority and decomposition rules. Also has a method (matches) which checks if an input sentence contains the keyword and if there is a matching decomposition rule for the sentence.	- The decomposition rules are stored in an ArrayList. -
eliza.wrapper.StringArrayRandomizer	This class stores a list of strings and returns random elements from it, never returning the same element twice until all elements have been returned. It is used to randomize the order in which reassembly rules and other responses are chosen.	- An ArrayList is used to store the information. -
eliza.wrapper.SubstitutionRule	Wrapper for pre- and post-substitution rules.	

Data representation

When considering how data should be represented efficiency and ease of understanding/user friendliness must be taken into consideration.

ArrayList vs HashMap for representing keywords

In order to make manipulation of the script as easy as possible, we chose to use an ArrayList rather than a HashMap to represent the keywords. This decision meant that the keywords could be sorted after being read instead of having to be sorted in the script file for maximum efficiency. A HashMap would also be a valid solution, but we would have to sort the script file before being read. Otherwise the list of possible keywords could be longer.

ArrayList vs Array

The main reason ArrayLists were used was because the number of elements in arrays could not be pre-determined easily. Converting to normal arrays was a possibility, but would not have a substantial effect on performance and would make initialization of the conversation take longer.

File parsing

Reading the file

A BufferedReader was used to read the script file because it improves efficiency of how the data is accessed, through use of a buffer which read large chunks of the file into RAM for quicker access. A Scanner could have been used but since it was not required to parse many integers or tokens it was simpler to use a BufferedReader.

Compiling Patterns

Patterns are part of regular expressions in Java, and it would have been possible to reformat the decomposition rules and compile them whilst reading the file. The Patterns were compiled while decomposing sentences instead of whilst reading the file, because it made debugging easier.

Textual manipulation

Although the engine could be successfully implemented without use of regular expressions, they make textual manipulation much simpler. An alternative would be to use String methods such as substring, indexOf, charAt to manipulate the text, but this would have been more complicated.

Sorting algorithm: Quicksort

A quicksort algorithm was used to sort the Arraylist of Keywords by ascending priority. This decision was made because it is a nlog(n)[1] sorting algorithm and, on average, uses fewer comparisons than most sorting algorithms. Possible alternatives could be a heap-, merge-, or quick-sort median-of-3 – algorithms.

[1] http://www.sorting-algorithms.com/quick-sort

Limitations

Engine

If the user enters a proper noun Eliza will de-capitalize it if it repeats it. Example:

```
Eliza: Hello, I'm Eliza. What would you like to discuss?
>>Hi Eliza. I want to talk about Bill Gates.
Eliza: What makes you interested in bill gates?
```

Syntax errors in the script file will not always be detected by the parser.

Example 1

If the decomposition rules are in the wrong order, one can override the other making it unreachable. Example:

```
k:problem 3
d:My problem *
r:Tell me more about 1.
r:Tell me about your problem.
d:My problem is *
r:Tell me more about the problem.
r:Please elaborate on 1.
r:Why do you think your problem is 1?
```

The second decomposition rule will never be used as the Engine will find the first decomposition before it finds the more detailed second one.

Example 2

```
k:a p1
k:b p2
```

If two keywords come one after the other, this will not be detected as an error. The same applies if two decomposition rules come one after the other.

```
d:x
d:y
```

Also, Eliza is unable to pick up on when the user contradicts itself:

```
Eliza: Hello, I'm Eliza. What would you like to discuss?
>>I would like to discuss my problem.
Eliza: Please elaborate.
>>I drink a lot
Eliza: You need to tell me more.
>>I come home from work every day, and down a bottle of jack.
... Later in the conversation:
>>I dropped out of high school early and have never had a job
Eliza: Please tell me more.
>>I am a layabout.
Eliza: What makes you a layabout?
>>I just sit around all day and go to work and don't know how to interact with people.
```

Not only does the user contradict them self once, by first saying he has a job and then he doesn't. He then contradicts himself again and says that he does in fact have a job. In normal conversation you would expect anybody to pick up on this but because of the way the script and engine are designed this is not possible.

Script

Eliza thinks almost any use of the word "you" is referring to her directly even though it is quite obviously not in the example below.

```
...
Eliza: I need you to tell me more.
>>Don't worry, it is a legal drug.
Eliza: Please go into more detail.
>>Coffee keeps you awake. It is a stimulant.
Eliza: I'm not going to talk about myself. Let's talk about you.
```

It is also unable to talk about itself in any detail and dodges almost all questions the user asks about it.

At the beginning of the conversation, the user could input something that contains no keywords. If this happens Eliza would have to use on the other-responses. The problem with this is that Eliza could instantly ask to talk about something else

```
Eliza: Hi, I'm Eliza. What do you want to talk about?
>>My grandmother
Eliza: Let's discuss something else.
```

The user would expect Eliza to engage in conversation about their grandmother and most of the time it would, but since “Let’s discuss something else” is an other-response, there is a chance Eliza will respond by asking to discuss something different. Unfortunatly it is necessary to have “Let’s discuss something else” as an other-response because it can also lead a dead conversation in a new direction.

```
>>I have my own rehearsal space.
Eliza: Do you know anyone else who has your own rehearsal space.
```

As seen above, the post-substitution rules in conjunction with the keywords will not always keep the meaning the same.

Testing

If the user enters an incredibly long string (Over 40530 characters in length) the program breaks and two user input lines appear but only one is usable. Since there is very almost no chance of the user entering a string of that length in normal use, validation to check the length of the string is not required. Example:

```
>>*VERY LONG STRING*
Eliza: I need you to elaborate on that
>>*OTHER VERY LONG STRING*
>>
Eliza: Please go into more detail. (It becomes impossible to highlight
parts of this line in the Eclipse Console now.)
>>g
Eliza: You need to elaborate on that.
>>*ANOTHER VERY LONG STRING*
>>that good?
Eliza: Can you elaborate on that?
>>k?
Eliza: Can you tell me more?
>>why are there two sets of arros?
Eliza: I know why, but what do you think?
>>£
Eliza: Please tell me more.
```

When entering very long strings into the Eclipse IDE Console, the program started to function incorrectly, giving two sets of arrows and making it impossible to highlight parts of the output. It also removes any previous console output from the console. This may be a problem with Java, the Eclipse IDE or the program itself. However in normal use it is unlikely the user would ever enter a string of that length so this is not too serious a problem.

Conclusion

In this project we created our own Eliza, with its own script syntax, engine and script. The original Eliza "Doctor" script simulated a conversation with a psychiatrist and that was what we also intended to do. Our script, however, is more like general chat with a person who has psychiatrist tendencies. In order to make the conversation seem more realistic, the original Eliza had a form of memory, which would repeat something said in an earlier input message when it could not locate any keywords in an input message. This would be something we would have added if more time was available for the project.

In addition to some sort of memory, we would have used a scanner to search through input sentences for keywords, which would be represented in a HashMap, allowing the engine to locate keywords faster and thus conversation output to be generated more quickly. Generally, though, we are quite happy with our result, and do not feel that the few milliseconds it takes the engine to decompose and reassembly sentences is an issue. Another feature we could have added would be a library of scripts with separate "personalities"/purposes. I.e. start the conversation with "Who do you want to talk to today?" and then let the user chose from a list.

References

1. Keyword-"believe", reassembly rule "Have you always believed 1?" from **CS1006 Lecture 1 Slide 38**
2. Keyword- "because" reassembly rule from, **http://chayden.net/eliza/instructions.txt**
3. Other Response- "Can you elaborate on that"from **CS1006 Lecture 1 Slide 35.**