Name: Anika Jallipalli  
Date: 1/24/2020

Magpie Chatbot Lab

**Directions**: Make note of your responses to the following questions as you work through activities 1, 2, and 3 of the AP Computer Science Lab: Magpie.

# Activity 1

1. Which chatbot did you use?

iEinstein

1. Paste your conversation with the chatbot below.

Me: Hi

Einstein: Hello! What’s your name?

Me: why?

Einstein: You can make up a name if you like. It just helps to make the conversation flow if I know what to call you. What's your name?

Me: How old are you?

Einstein: You're not paying attention. What's your name?

Me: Amy

Einstein: Amy is one of my chatbot friends.

Me: cool

Einstein: thanks

Me: Bye

Einstein: Until next time.

1. What was the most interesting or peculiar response? Why?

The most interesting thing is that the chatbox persisted to know my name, I thought that if there was no input then the chatbox would ignore it and move on.

1. Think about your conversation with a chatbot. Identify key words to which you think the chatbot responds.

Single word responses were most prominent, age, name and basic information, cant carry on a conversation, reads based on one response

1. Suggest several key words and the possible corresponding responses.

Hi – hello

Your name? – Einstein

Age- doesn’t reveal

How do you help- doesn’t know how to respond

# Activity 2

1. What happens when a key word is included in another word? Consider statements like "I know all the state capitals" and "I like vegetables smothered in cheese." Explain the problem with the responses.

They are robotic responses, not humanly. I like cheese responded with I understand what cheese is. The robot doesn’t have a response to counter the question.

1. The lab suggests using the trim method from the String class. How does it work? Show an example of a String value before and after the trim method is invoked.

The method returns a new string which is a exact value of the original string stripped with leading and trailing spaces

Hi

Hi

1. Paste the code below showing the additional two noncommittal responses added to getRandomResponse.

public String getResponse(String statement)

{

String response = "";

if (statement.indexOf("no") >= 0)

{

response = "Why so negative?";

}

else if (statement.indexOf("mother") >= 0

|| statement.indexOf("father") >= 0

|| statement.indexOf("sister") >= 0

|| statement.indexOf("brother") >= 0)

{

response = "Tell me more about your family.";

}

else

{

response = getRandomResponse();

}

return response;

}

/\*\*

\* Pick a default response to use if nothing else fits.

\* @return a non-committal string

\*/

private String getRandomResponse()

{

final int NUMBER\_OF\_RESPONSES = 4;

double r = Math.random();

int whichResponse = (int)(r \* NUMBER\_OF\_RESPONSES);

String response = "";

if (whichResponse == 0)

{

response = "Interesting, tell me more.";

}

else if (whichResponse == 1)

{

response = "Hmmm.";

}

else if (whichResponse == 2)

{

response = "Do you really think so?";

}

else if (whichResponse == 3)

{

response = "You don't say.";

}

return response;

1. Complete the table below. List the additional keywords and responses you added to the getResponse method.

| Keyword | Response |
| --- | --- |
| I want to | "What would it mean to |
| I want | "Would you really be happy if you had |
| you <something> me | "What makes you think that I <something> you?" |

1. What happens when more than one keyword appears in a string? Consider the string "My mother has a dog but no cat." Explain how to prioritize responses in the reply method. Did this impact any changes you made to the getResponse method?

# Activity 3

Trace through the following method calls. Write the value of the variables position, before, and after each time the program control reaches the point in the method indicated by the comment.

1. findKeyword( "She’s my sister", "sister", 0);

| Iteration | position | before | after |
| --- | --- | --- | --- |
| 1 | 9 | “ “ | “ “ |

1. findKeyword( "Brother Tom is helpful", "brother", 0);

| Iteration | position | before | after |
| --- | --- | --- | --- |
| 1 | 0 | “ “ | “ “ |

1. findKeyword( "I can’t catch wild cats.", "cat", 0);

| Iteration | position | before | after |
| --- | --- | --- | --- |
| 1 | 8 | “ “ | “c” |
| 2 | 19 | “ “ | “S” |
| 3 | -1 |  |  |

1. findKeyword( "I know nothing about snow plows.", "no", 0);

| Iteration | position | before | after |
| --- | --- | --- | --- |
| 1 | 3 | “K” | “w” |
| 2 | 7 | “ “ | “t” |
| 3 | 22 | “s” | “w” |
| 4 | -1 |  |  |

1. Your choice. Fill in the parameters and then trace the method call.  
   findKeyword( "", "", );

| Iteration | position | before | after |
| --- | --- | --- | --- |
| 1 | 3 | “K” | “w” |
| 2 | 7 | “ “ | “t” |
| 3 | 22 | “s” | “w” |
| 4 | -1 |  |  |

PMR: This program was so hard to use. This entire lesson was useless, and the worksheet was dumb. I was so confused the entire time. I didn’t find anything easy about this lesson. I am disappointed.

**MAGPIE2**

/\*\*

\* A program to carry on conversations with a human user.

\* This is the initial version that:

\* <ul><li>

\* Uses indexOf to find strings

\* </li><li>

\* Handles responding to simple words and phrases

\* </li></ul>

\* This version uses a nested if to handle default responses.

\* @author Anika Jallipalli

\* @version 1/24/2020

\*/

public class Magpie2

{

/\*\*

\* Get a default greeting

\* @return a greeting

\*/

public String getGreeting()

{

return "Hello, let's talk.";

}

/\*\*

\* Gives a response to a user statement

\*

\* @param statement

\* the user statement

\* @return a response based on the rules given

\*/

public String getResponse(String statement)

{

String response = "";

if (statement.indexOf("no") >= 0)

{

response = "Why so negative?";

}

else if (statement.indexOf("mother") >= 0

|| statement.indexOf("father") >= 0

|| statement.indexOf("sister") >= 0

|| statement.indexOf("brother") >= 0)

{

response = "Tell me more about your family.";

}

else

{

response = getRandomResponse();

}

return response;

}

/\*\*

\* Pick a default response to use if nothing else fits.

\* @return a non-committal string

\*/

private String getRandomResponse()

{

final int NUMBER\_OF\_RESPONSES = 4;

double r = Math.random();

int whichResponse = (int)(r \* NUMBER\_OF\_RESPONSES);

String response = "";

if (whichResponse == 0)

{

response = "Interesting, tell me more.";

}

else if (whichResponse == 1)

{

response = "Hmmm.";

}

else if (whichResponse == 2)

{

response = "Do you really think so?";

}

else if (whichResponse == 3)

{

response = "You don't say.";

}

return response;

}

/\*\*

\* Search for one word in phrase. The search is not case

\* sensitive. This method will check that the given goal

\* is not a substring of a longer string (so, for

\* example, "I know" does not contain "no").

\*

\* @param statement

\* the string to search

\* @param goal

\* the string to search for

\* @param startPos

\* the character of the string to begin the

\* search at

\* @return the index of the first occurrence of goal in

\* statement or -1 if it's not found

\*/

private int findKeyword(String statement, String goal,

int startPos)

{

String phrase = statement.trim();

// The only change to incorporate the startPos is in

// the line below

int psn = phrase.toLowerCase().indexOf(

goal.toLowerCase(), startPos);

// Refinement--make sure the goal isn't part of a

// word

while (psn >= 0)

{

// Find the string of length 1 before and after

// the word

String before = " ", after = " ";

if (psn > 0)

{

before = phrase.substring(psn - 1, psn)

.toLowerCase();

}

if (psn + goal.length() < phrase.length())

{

after = phrase.substring(

psn + goal.length(),

psn + goal.length() + 1)

.toLowerCase();

}

// If before and after aren't letters, we've

// found the word

if (((before.compareTo("a") < 0) || (before

.compareTo("z") > 0)) // before is not a

// letter

&& ((after.compareTo("a") < 0) || (after

.compareTo("z") > 0)))

{

return psn;

}

// The last position didn't work, so let's find

// the next, if there is one.

psn = phrase.indexOf(goal.toLowerCase(),

psn + 1);

}

return -1;

}

/\*\*

\* Search for one word in phrase. The search is not case

\* sensitive. This method will check that the given goal

\* is not a substring of a longer string (so, for

\* example, "I know" does not contain "no"). The search

\* begins at the beginning of the string.

\*

\* @param statement

\* the string to search

\* @param goal

\* the string to search for

\* @return the index of the first occurrence of goal in

\* statement or -1 if it's not found

\*/

private int findKeyword(String statement, String goal)

{

return findKeyword(statement, goal, 0);

}

}

**MAGPIERUNNER2**

import java.util.Scanner;

/\*\*

\* A simple class to run the Magpie class.

\* @author Anika Jallipalli

\* @version 1/24/2020

\*/

public class MagpieRunner2

{

/\*\*

\* Create a Magpie, give it user input, and print its replies.

\*/

public static void main(String[] args)

{

Magpie2 maggie = new Magpie2();

System.out.println (maggie.getGreeting());

Scanner in = new Scanner (System.in);

String statement = in.nextLine();

while (!statement.equals("Bye"))

{

System.out.println (maggie.getResponse(statement));

statement = in.nextLine();

}

}

}

**STRINGEXPLORER**

/\*\*

\* A program to allow students to try out different

\* String methods.

\* @author Anika Jallipalli

\* @version 1/24/2020

\*/

public class StringExplorer

{

public static void main(String[] args)

{

String sample = "The quick brown fox jumped over the lazy dog.";

// Demonstrate the indexOf method.

int position = sample.indexOf("quick");

System.out.println ("sample.indexOf(\"quick\") = " + position);

// Demonstrate the toLowerCase method.

String lowerCase = sample.toLowerCase();

System.out.println ("sample.toLowerCase() = " + lowerCase);

System.out.println ("After toLowerCase(), sample = " + sample);

// Try other methods here:

}

}