AP Computer Science

AP Java Coding Labs [Ver. 2.0]

Lab B1: Magpie Lab

WEEK 1:

DR. ERIC CHOU

IEEE SENIOR MEMBER

Overview

LECTURE 1

ACTIVITY 1: Introduction

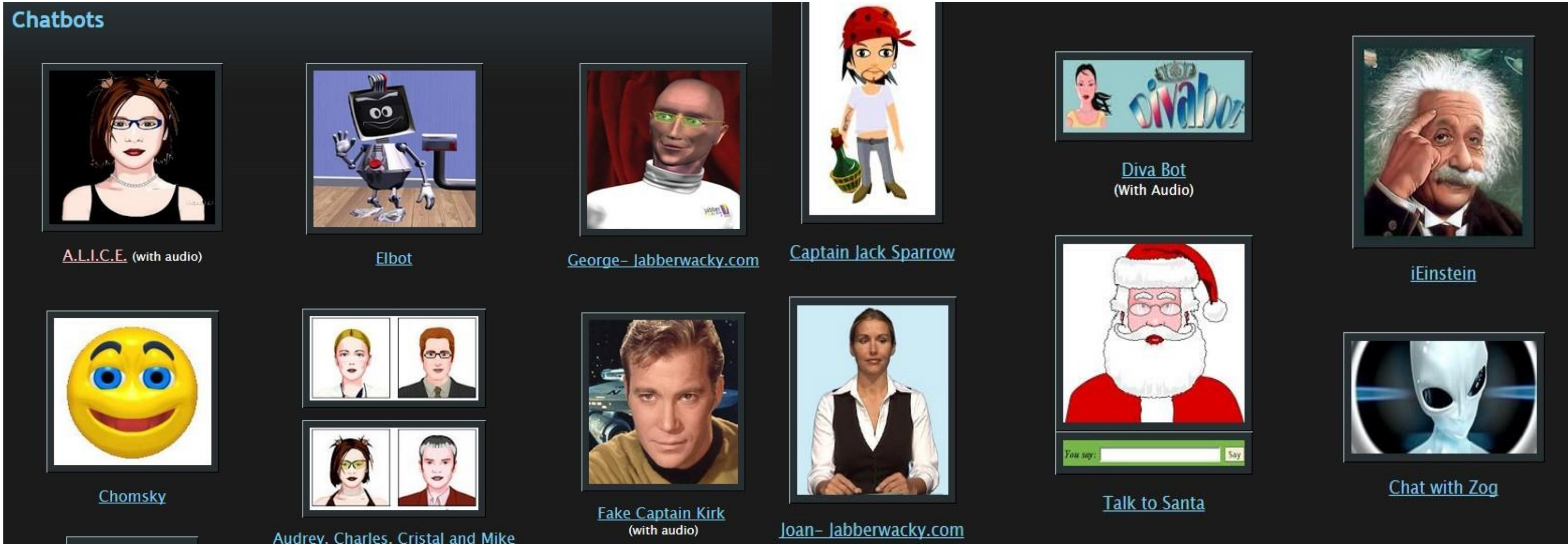
LECTURE 1

Activity One:

- Getting Acquainted with Chatbots
- Chatbots are programs that are design to respond like humans to natural language input. Before you write code to create you own chatbot, you will explore a list of chatbots for you to try.
- Start
- Go to http://sites.google.com/site/webtoolsbox/bots
- Exploration:
- Have several conversations with your chatbot and answer the following questions:
  - How does it respond to “where do you come from?”
  - What is the most interesting response?
  - What is the most peculiar response?
  - How does it respond to “asdfghjkl;”?

Experience Chat bots

http://sites.google.com/site/webtoolsbox/bots



Chatbots

A.L.I.C.E. (with audio)

Elbot

George– Jabberwacky.com

Captain Jack Sparrow

Diva Bot
(With Audio)

iEinstein

Chomsky

Audrey, Charles, Cristal and Mike

Fake Captain Kirk
(with audio)

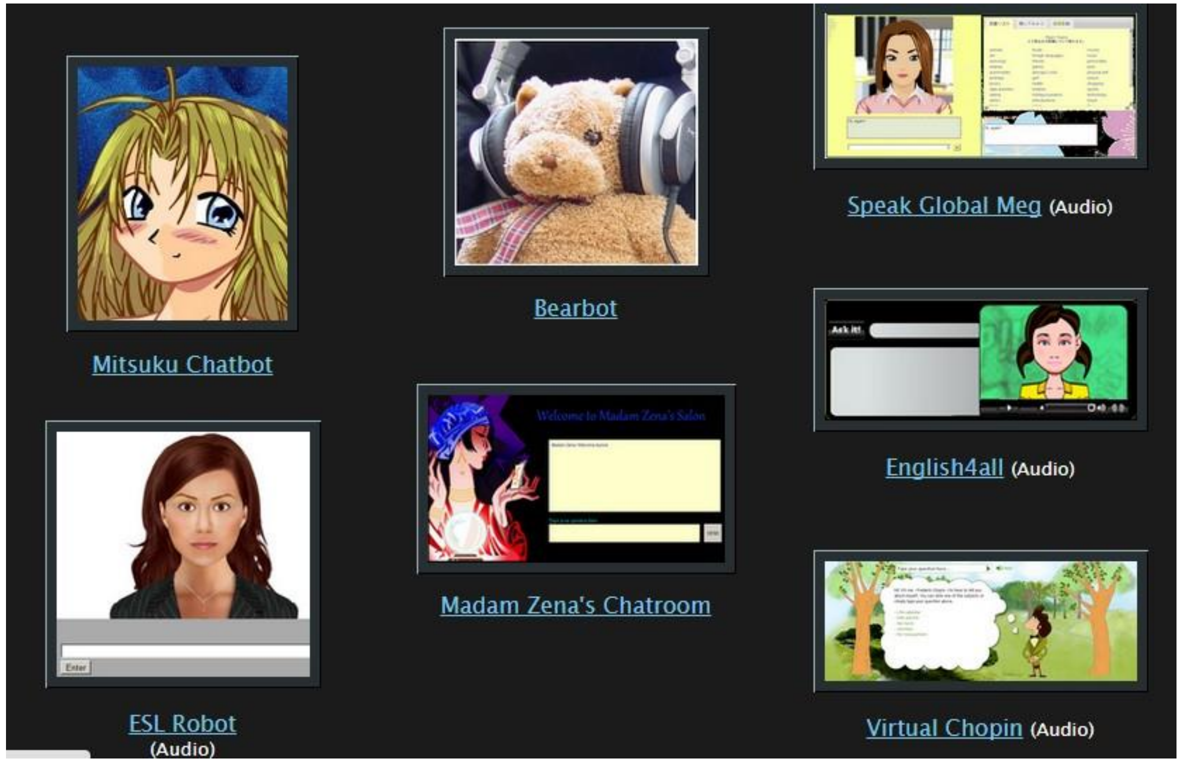
Joan– Jabberwacky.com

Talk to Santa

Chat with Zog

Experience Chat bots

http://sites.google.com/site/webtoolsbox/bots


Mitsuku Chatbot

Bearbot

Speak Global Meg (Audio)

English4all (Audio)

ESL Robot (Audio)

Madam Zena's Chatroom

Virtual Chopin (Audio)

Questions

Simple chatbots act by looking for the key words or phrases and responding to them.

1. Can you identify keywords to which your chatbot responds?

2. Think of several keywords and the responses they might cause.

Activity Two:
Introduction to the Magpie Class (I)

In this activity, you will work Magpie, with simple implementation of a chatbot. You will see how it works with some keywords and add keywords of your own.

Prepare:

Have available:

- The code for the Magpie
- The code for the MagpieRunner
- A computer with your Java development tools

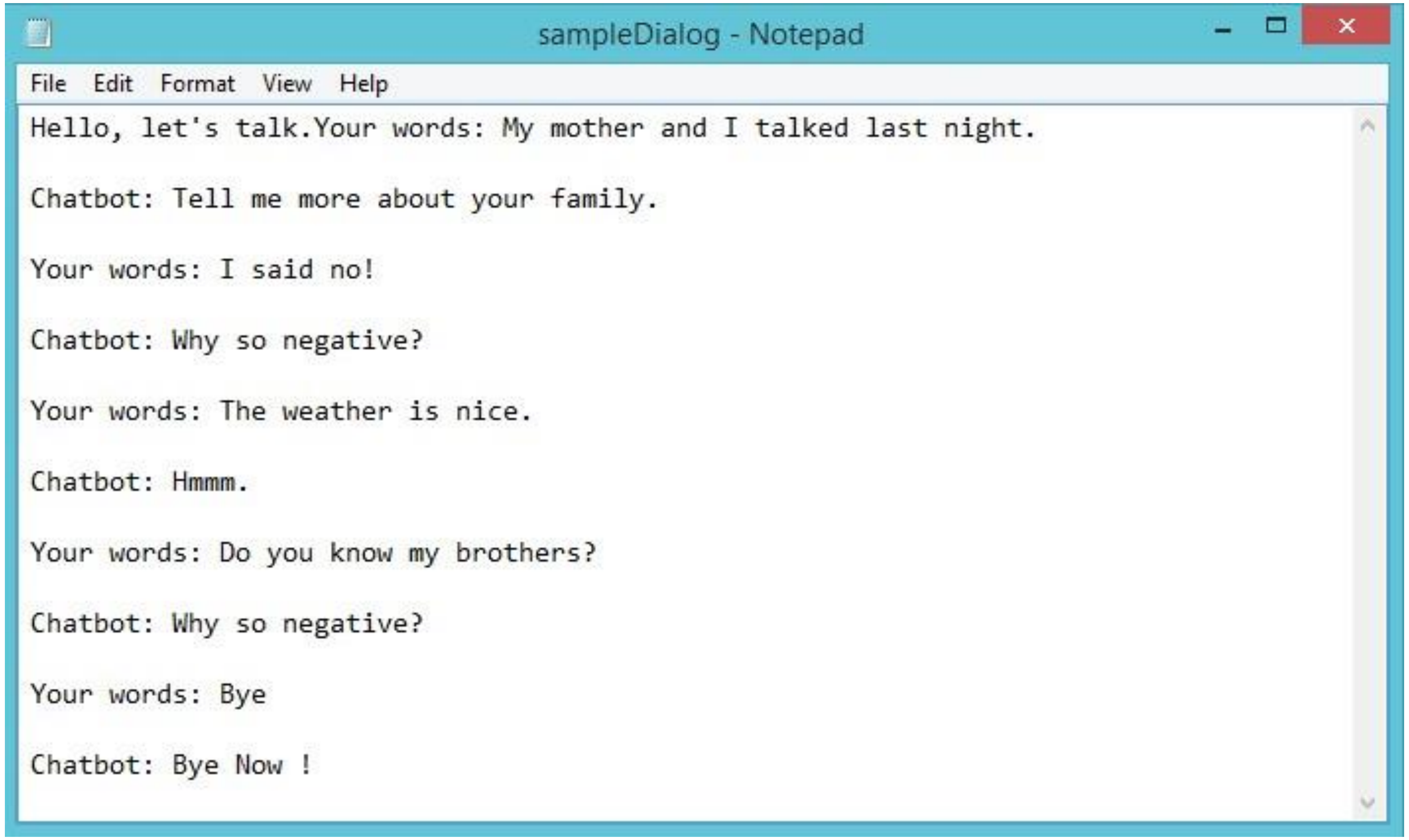
Activity Two: Introduction to the Magpie Class (II)

Start:

How does it responds to:

- My mother and I talked last night.
- I said no!
- The weather is nice.
- Do you know my brother?

Initial Part Results:
Magpie2 and MagpieRunner2



```
Hello, let's talk.Your words: My mother and I talked last night.

Chatbot: Tell me more about your family.

Your words: I said no!

Chatbot: Why so negative?

Your words: The weather is nice.

Chatbot: Hmmm.

Your words: Do you know my brothers?

Chatbot: Why so negative?

Your words: Bye

Chatbot: Bye Now !
```

Activity Two: Introduction to the Magpie Class (III)

Exploration:

Look at the code. See how the if statement assigns a value to the response and returns that response. The method **getRandomResponse picks a response from a group of** String objects.

Exploration:

Alter the code:

- Have it respond “Tell me more about your pets” when the statement contains the word “dog: or “cat.” For example, a possible statement and response would be:

  Statement 1: like my cat Mittens.

  Response: Tell me more about your pets.

Try your own design !!

Example:

```
else if (statement.indexOf("dog") >= 0
            || statement.indexOf("cat") >= 0
            || statement.indexOf("rabbit") >= 0
            || statement.indexOf("fish") >= 0
            )
{
      response = "Tell me more about your pets.";
}  // add this code after the else if for family
```

Activity Two: Introduction to the Magpie Class (IV)

- Have it respond favorably when it sees the same of your teacher. Be sure to use appropriate pronouns! For example, a possible statement and response would be:

  Statement: Mr. Finkelstein telling us about robotics.

  Response: He sounds like a good teacher.

- Have the code check that the statement has at least one character. You can do this by using the trim method to remove spaces from the beginning and end, and then checking the length of the trimming string. If there are no characters, the response should tell the user to enter something. For example, a possible statement and response would be:

  Statement:

  Response: Say Something Please.

Teacher Identification

Example:

```
else if (statement.indexOf("Mr. Chou") >= 0  // Identiry Teacher
                    || statement.indexOf("Mr. Ho") >= 0
                    || statement.indexOf("Mr. Lee") >= 0
                    || statement.indexOf("Mr. Robinson") >= 0
                    )
        {
                response = "He sounds like a good teacher.";
        }
```

Handler for the Empty Response

Example:

```
else if ( statement.trim().equals(“”) )   // No response from user
    {
        response = “Say Somehting Please !”;
    }
```

ACTIVITY 2: Keyword and Response

LECTURE 1

Activity Two: Introduction to the Magpie Class (V)

- Add two more noncommittal responses to the possible random responses.
- Pick three more keywords, such as “no” and “brother” and edit the getResponse method to respond to each of these. Enter three keywords and responses below.

Keyword	Response

- What happens when more than one keyword appears in a string? Consider the string “My mother has a dog but not cat.” Explain how to prioritize responses in the reply method.

Question

1. What happens when a keyword 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 response to these statements.

Add two more noncommittal responses to the possible random responses.

Update the NUMBER_OF_RESPONSES:

```
        final int NUMBER_OF_RESPONSES = 6;
```

Example (Two example):

```
          else if (whichResponse == 4)
              {
                      response = "I don't quite understand.";
              }
              else if (whichResponse == 5)
              {
                      response = "Try something different. ";
              }
```

Keyword::Response Pair

Keyword	Response
lunch	Are you hungry?
sun	It is a beautiful day.
long	Are you measuring something?

```
else if (statement.indexOf("lunch") >= 0 ) // Lunch
    {
        response = "Are you hungry?";
    }
    else if (statement.indexOf("sun") >= 0 ) // Sun
    {
        response = "It is a beautiful day.";
    }
else if (statement.indexOf("long") >= 0 ) // long
    {
        response = "Are you measuring something?";
    }
```

What happens when more than one keyword appears in a string?

The current programming style is exclusive:

```
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.";
}
```

Change to non-exclusive:

```
        if (statement.indexOf("no") >= 0)
{
        response = "Why so negative?";
}
if (statement.indexOf("mother") >= 0
                || statement.indexOf("father") >= 0
                || statement.indexOf("sister") >= 0
                || statement.indexOf("brother") >= 0
                )
{
        response = "Tell me more about your family.";
}
```

Then, if keyword in group one and group two show up in the same sentence, both will get responded.

Using 2-D Array to create keyword and Response pairs

Create a big 2-D Keyword::Response pair Dictionary:

```
String[][] keyResp = {
            {"dinner", "I am so hungry."},
            {"tired", "Get some rest!"},
            {"money", "What do you want to buy"},
           };
```

Simple non-exclusive response engine:

```
        for (int i=0; i<keyResp.length; i++)
          {  if (statement.indexOf(keyResp[i][0]) >= 0 ) // Sun
             {
                  response = keyResp[i][1];
             }
          }
```

ACTIVITY 3: Better Keyword Detection (I)

LECTURE 1

Activity Three: Better Keyword Detection (I)

In the previous activity, you discovered that simply searching for a collections of letters in a string does not always work as intended. For example, the word “cat” in the string “Let’s play catch !,” but the string has nothing to do with the animal. In this activity, you will trace a method that searches for a full word in the string. It will check the substring before and after the string to ensure that the keyword is actually found.

You will use some more complex **String** methods in this activity. The **String** class has many useful methods, not all of which are included in the AP Computer Science Java Subset. But they can be helpful in certain cases, so you will learn how to use the API to explore all of the methods that are built into Java.

Activity Three:
Better Keyword Detection (II)

Prepare:

Have Available:

The API for the Magpie class

The API for the String class

The code for the StringExplorer

The code for the Magpie

The code for the MagpieRunner

A computer with your Java development tools.

Activity Three:
Better Keyword Detection (III)

Exploration: Using the API

One of the major benefits of using **Java** as a programming language is that so many library already been created for it.

Open the program **StringExplorer**. It currently has code to illustrate the use of **indexOf** and **toLowerCase** methods.

Open the API for **String**. Scroll down to the Method Summary section and find the **indexOf(String str)** method. Follow the link and read the description of the method. What value is returned by **indexOf** if the substring does not occur in the string?

Add the following lines to **StringExplorer** to see for yourself that **indexOf** behaves as specified:

int notFoundPsn = sample.indexOf("slow");

Activity Three: Better Keyword Detection (IV)

System.out.println("sample.indexOf(\"slow\”) = “ + notFoundPsn);

Read the description of **indexOf(String str, int fromIndex).** Add lines to **StringExplorer** that illustrate how this version of **indexOf** differs from the one with one parameter.

This lab activity will use a variety of different String methods. Consult the **API** whenever you see one with which you are unfamiliar.

Activity Three: Better Keyword Detection (V)

This version of the **Magpie** class has a method named **findKeyword** to detect keywords. This method will only find exact matches of the keyword, instead of cases where the keyword is embedded in a longer word. Run it, using the instructions provided by your teacher.

```
private int findKeyword(String statement, String goal, int startPos) {
      String phrase = statement.trim().toLowerCase();
      goal = goal.toLowerCase();
      int psn = phrase.indexOf(goal, startPos);  // index first found goal

     while (psn >= 0)  {   // goal is part of statement
            String before = " ", after = " ";
           if (psn > 0) {
               before = phrase.substring(psn – 1, psn);  // find before charater
           }
           if (psn + goal.length() < phrase.length()) { // find after character
               after=phrase.substring( psn + goal.length(),
                                                   psn + goal.length() + 1);
           }
```

Activity Three: Better Keyword Detection (VI)

```
     /* determine the values of psn, before, and after at this point in the method. */
     if (   (( before.compareTo ("a") < 0 ) || (before.compareTo("z") >0) ) &&
             ((after.compareTo ("a") < 0 ) || (after.compareTo("z") > 0))   ) {  // before and after are alphabet letters
            return psn;   // return the index location for goal which is a whole word in statement.
     }
     psn = phrase.indexOf(goal, psn + 1);   // find next candidate
   }
   return -1;   // goal not in statement or not a whole word
}
```

string.compareTo(“a”);

The result of this compareTo function is

ASCII(string) – ASCII(“a”);

Reversed Engineered pseudo code for findKeyword(statement, goal)

```
(1) Covert the statement and goal to lowercase.
(2) Find the index of the first occurrence of goal string.
(3)  while (still can find the next occurrence of goal string) {
         set the initial assignment of before and after to space " " letter;
         if (goal still exist in statement and not at the beginning of statement) {
                 set before to the letter before the current occurrence;
            }
         if (ending letter of goal still in statement) {
                 set after to the end of current goal string;
            }
         if (both before and after are not in alphabet) {
                     return current occurrence index;
            }
        otherwise, find the next occurrence;
      }
(4) If there is no legal index returned, return -1 to signal for goal not existing  in statement.
```

Activity Three: Better Keyword Detection (VII)

Read through the **findKeyword** method. To ensure that you understand it, trace the following method calls.

```
findKeyword("She's my sister", "sister", 0);
findKeyword("Brother Tom is helpful", "brother", 0);
findKeyword("I can't catch wild cats.", "cat", 0);
findKeyword("I know nothing about snow plows.", "no", 0);
```

Write the value of each of the variable **psn**, **before**, and **after** each time the program control reaches the point in the method indicated by the comment.

Activity Three: Better Keyword Detection (VIII)

Example: **findKeyword(“yesterday is today’s day before.”, “no”, 0);**

Iteration	psn	before	after
1	6	“r”	“ “
2	15	“o”	“ ’ ”
3	21	“ “	“ “

Activity Three: Better Keyword Detection (IX)

Use a copy of the table below to trace the calls.

Iteration	psn	before	after

Activity Three:
Better Keyword Detection (X)

Exercise: Use the new method

Repeat the changes you made to the program in Activity 2, using the new method to detect keywords.

Questions: Prepare for the next activity

Single keywords are interesting, but better chatbots look for groups of words. Consider statements like “I like cats,” “I like math class,” and “I like Spain.” All of these have the form “I like something.” The response could be “What do you like about something?” The next activity will expand on these groups. You will get to add one of your own, so it’s a good idea to start paying close attention to common phrases in your own conversations.

Example: Replace statement.indexOf(“sun”); to findKeyword(statement, “sun”);

```
// Example to use 2-D array for keyword:response pair
        String[][] keyResp = {
          {"dinner", "I am so hungry."},
          {"tired", "Get some rest!"},
          {"money", "What do you want to buy"},
         };
        for (int i=0; i<keyResp.length; i++)
          {  if (findKeyword(statement, keyResp[i][0]) >= 0 ) // Check key
               {
                    response = keyResp[i][1];
               }
          }
```

ACTIVITY 4: Responses that Transform Statements

LECTURE 1

Activity Four: Responses that Transform Statements (I)

As stated previously, single keywords are interesting, but better chatbots look for groups of words. Statements like “I like cats,” “I like math class,”and “I like Spain” all have the form “I like something.” The response could be “What do you like about something?” This activity will respond to groupings of words.

Prepare

Have available:

- the API for the Magpie class
- the API for the String class
- the code for the Magpie
- the code for the MagpieRunner
- a computer with your Java development tools

Activity Four: Responses that Transform Statements (II)

Exploration

Get to know the revised Magpie class. Run it, using the instructions provided by your teacher.
How does it respond to:

- I want to build a robot.
- I want to understand French.
- Do you like me?
- You confuse me.

Exercises

Look at the code. See how it handles “I want to” and you/me statements.
Alter the code:

- Have it respond to “I want something” statements with “Would you really be happy if you had something?” In doing this, you need to be careful about where you place the check. Be sure you understand why. For example:

Statement: I want fried chicken.

Response: Would you really be happy if you had fried chicken?

Activity Four: Responses that Transform Statements (II)

```
    if (findKeyword(statement, "I want to", 0) >= 0)
{  response = transformIWantToStatement(statement);
}
else
{       // Look for a two word (you <something> me)
        // pattern
        int psn = findKeyword(statement, "you", 0);
        if (psn >= 0 && findKeyword(statement, "me", psn) >= 0)
        {    response = transformYouMeStatement(statement);
        }
        else
        {   response = getRandomResponse();
        }
}
```

Qualified with “I want to” case, do transformation and get results.
Qualified with “you” and “me” case, do another transformation and get results.

```
Hello, let's talk.
Your words: I want to go to school
What would it mean to go to school?
Your words: bye
Chatbot: Bye Now !
```

“I want to” Transform

```
private String transformIWantToStatement(String statement)
    {     //  Remove the final period, if there is one
          statement = statement.trim();
          String lastChar = statement.substring(statement.length() - 1);
          if (lastChar.equals("."))
          {
                statement = statement.substring(0, statement.length() - 1);
          }
          int psn = findKeyword (statement, "I want to", 0);
          String restOfStatement = statement.substring(psn + 9).trim();
          return "What would it mean to " + restOfStatement + "?";
    }
```

Techniques learned from this part.

(1) use findKeyword to locate a whole phrase “I want to”

(2) Take out part of a sentence. In this case, the program take out all of the words after the “I want to” using substring. Then, use trim() to take out extra space. Then, take out the period symbol.

(3) Insert this remaining part back to a new sentence using string concatenation.

Activity Four: Responses that Transform Statements (III)

- Have it respond to statements of the form “I something you” with the restructuring “Why do you something me?” For example:

Statement: I like you.

Response: Why do you like me?

Find an example of when this structure does not work well. How can you improve it?

Activity Four: Responses that Transform Statements (III)

```
private String transformYouMeStatement(String statement)
    {     //  Remove the final period, if there is one
          statement = statement.trim();
          String lastChar = statement.substring(statement.length() - 1);
          if (lastChar.equals("."))
          {   statement = statement.substring(0, statement.length() - 1);
          }

          int psnOfYou = findKeyword (statement, "you", 0);
          int psnOfMe = findKeyword (statement, "me", psnOfYou + 3);

          String restOfStatement = statement.substring(psnOfYou + 3, psnOfMe).trim();
          return "What makes you think that I " + restOfStatement + " you?";
    }
```

Techniques learned from this part.

(1) use findKeyword to locate a whole phrase “I” and “you”

(2) Take out part of a sentence between “I” and “you”. Locate the two strings. Extract the substring between them. Using the index to the end of “I” and the index before “you”.

(3) Insert this remaining part back to a new sentence using string concatenation.

ACTIVITY 5: Arrays and the Magpie

LECTURE 1

Activity Five: Arrays and the Magpie (I, Optional)

When you last worked with the Magpie, default responses were handled with a nested if statement. This certainly worked, and you could add more responses, but it was a bit awkward. An easier way to keep track of default responses is with an array. In this activity, you will see how an array makes handling default responses much easier.

Prepare

Have available:

- the code for the Magpie
- the code for the MagpieRunner
- a computer with your Java development tools

Activity Five: Arrays and the Magpie (II, Optional)

Exploration

Run this version of the Magpie class. You should see no difference in its outward behavior. Instead, it has been changed so that its internal structure is different. This is called code refactoring. That’s one of the big benefits of dealing with methods as black boxes. As long as they perform the action required, the user does not care about how they perform the action.

Read the code for getRandomResponse. Notice that it uses an array of responses.

Exercise

Alter the array to add four additional random responses. Notice that, because the getRandomResponse method uses the length attribute of the array, you do not need to change anything else.

Compile and run your code. You should run it until you see all of your new responses.

Current Work in NLP

There is much work going on with Natural Language Processing in a variety of areas:

•Spam filtering uses NLP to determine whether an email message is spam.

•Many businesses use virtual agents to provide assistance to customers on Websites.

•Information retrieval parses text, such as email messages, and tries to extract relevant information from it. For example, some email programs will suggest additions to online calendars based on text in the message.

•Sentiment analysis takes information retrieval further. Rather than extract information from a single source, it goes to a variety of online resources and accumulates information about a particular topic. For example, sentiment analysis might follow Twitter to see how people are reacting to a particular movie.

• Question answering systems search a large body of knowledge to respond to questions from users. The best known question answering system is IBM's Watson.

Glossary

API —An abbreviation for Application Programming Interface. It is a specification intended to be used as an interface by software components to communicate with each other.

Chatbot—A program that conducts a conversation with a human user.

Code refactoring —A disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

Magpie—Magpies are large black birds. They are thought to be among the most intelligent birds and are capable of mimicking human speech.

NLP—Natural Language Processing; the field that studies responding to, and processing, human language.

Virtual agent —Also known as an Intelligent Agent. This can be used to gather information from a customer so that appropriate action can be taken in response to a query.

References

General information

http://www.nlp-class.org. A free online NLP class from Stanford. The first video does a good job of explaining the problems and promises of NLP, although much of the material is at a much higher level.

http://nsf.gov/cise/csbytes/newsletter/vol1i4.html. An issue of the NSF Bits and Bytes Newsletter

dedicated to NLP. Professor Mari Ostendorf of the University of Washington is featured in the newsletter.

Some chatbots

http://www-03.ibm.com/innovation/us/watson/what-is-watson/science-behind-an-answer.html. Numerous videos about the creation of Watson.

http://nlp-addiction.com/chatbot. Versions of the Eliza program, the first widespread chatbot.

Women in NLP Research

http://dotdiva.org/profiles/laura.html. A virtual nurse created by Laura Pfeifer.

http://people.ischool.berkeley.edu/~hearst. Professor Marti Hearst's homepage. She researches user interfaces to search engines.