AP® Computer Science A

Magpie Chatbot Lab

Student Guide

The AP Program wishes to acknowledge and thank Laurie White of Mercer University, who developed this lab and the accompanying documentation.

CollegeBoard

Magpie Chatbot Lab: Student Guide

Introduction

From Eliza in the 1960s to Siri and Watson today, the idea of talking to computers in natural language has fascinated people. More and more, computer programs allow people to interact with them by typing English sentences. The field of computer science that addresses how computers can understand human language is called Natural Language Processing (NLP).

NLP is a field that attempts to have computers understand natural (i.e., human) language. There are many exciting breakthroughs in the field. While NLP is a complicated field, it is fairly easy to create a simple program to respond to English sentences.

For this lab, you will explore some of the basics of NLP. As you explore this, you will work with a variety of methods of the `String` class and practice using the `if` statement. You will trace a complicated method to find words in user input.

Activity 1: Getting Acquainted with Chatbots (Optional)

Chatbots are programs that are designed to respond like humans to natural language input. Before you write code to create your own chatbot, you will explore some existing chatbots.

Start

Go to h HYPERLINK "http://sites.google.com/site/webtoolsbox/bots" HYPERLINK "http://sites.google.com/site/webtoolsbox/bots" HYPERLINK "http://sites.google.com/site/webtoolsbox/bots"tout several of the chatbots and find one to use for this activity. Your teacher may have a specific list of chatbots for you to try.

Exploration

Have several conversations with your chatbot and answer the following questions:

- How does it respond to "where do you come from"?

The chat bot responds by saying "I'm from a planet in the Andromeda System."

- What is the most interesting response?

When it did not want to add one + one

- What is the most peculiar response?

The most interesting responce was when I responed with my ip address.

- How does it respond to "asdfghjkl;"?

The chat bot responed with "asdfghjkl? I don't understand."

Exercises

Work with another group and have two different chatbots converse with each other.

Questions

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

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

Some key words it can repond to is anything related to family, fanboys, the four w, along with many other geniral simple phrases.

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

If I were to start a question with what then the chatbot will respond in a answer type format.

Activity 2: Introduction to the `Magpie` Class

In this activity, you will work `Magpie`, with a 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

Start

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

- My mother and I talked last night.

Tell me more about your family.

- I said no!

Why so negative?

- The weather is nice.

Interesting, tell me more.

- Do you know my brother?

Why so negative?

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.

Exercises

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: I like my cat

  Mittens.

  Response: Tell me more about your pets.

- Have it respond favorably when it sees the name of your teacher. Be sure to

use appropriate pronouns! For example, a possible statement and response would be:

Statement: Mr. Finkelstein is 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 trimmed 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.

- 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 the three keywords and responses below.

Keyword	Response
no	nu
Mr.Holcomb	Mr.Holcomb != bad teacher
father	are you winning son

- 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.

It goes in order of the programed if statments because the program can not multithred. So what happens is that if I enter brother no it sees no first because it is ahead of brother in the if statmens.

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 responses to these statements.

It will take these staments as it it were still containing the key word because the `indexOf()` method takes what ever is in the string regardless if there is anything surrounding the key word.

Activity 3: Better Keyword Detection

In the previous activity, you discovered that simply searching for collections of letters in a string does not always work as intended. For example, the word "cat" is 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.

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

Exploration: Using the API

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

Open the program `StringExplorer`. It currently has code to illustrate the use of the `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 `indexOf` 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");

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.

Exploration: Understand the new method

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);

  while (psn >= 0)
  {
    String before = " ", after = " ";
    if (psn > 0)
    {
      before = phrase.substring(psn - 1, psn);
    }
    if (psn + goal.length() < phrase.length())
    {
      after = phrase.substring(psn + goal.length(),
                               psn + goal.length() + 1);
    }
    /* 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)))
    {
      return psn;
    }
    psn = phrase.indexOf(goal, psn + 1);
  }

  return -1;
}
```

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.

Example: `findKeyword("yesterday is today's day before.", "day", 0);`

Iteration	psn	before	after
1	6	"r"	" "
2	15	"o"	"'"
3	21	" "	" "

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

Iteration	`psn`	`before`	`after`
1	9	""	""
1	0	""	""
1	8	""	"c"
2	19	""	"s"

Exercise: Use the new method

Repeat the changes you made to the program in Activity 2, using this 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.

Activity 4: Responses that Transform Statements

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

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.

What would it mean to build a robot?

- I want to understand French.

What would it mean to understand French?

- Do you like me?

What makes you think that I like you?

- You confuse me.

What makes you think that I confuse you?

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?

- 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?

Statement: ym

Response: Why do you ym me?

Activity 5: Arrays and the Magpie (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

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 Web sites.

- 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

h HYPERLINK "http://www.nlp-class.org/" HYPERLINK "http://www.nlp-class.org/" HYPERLINK "http://www.nlp-class.org/"tt HYPERLINK "http://www.nlp-class.org/" HYPERLINK "http://www.nlp-class.org/" HYPERLINK "http://www.nlp-class.org/"pA 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.

h HYPERLINK "http://nsf.gov/cise/csbytes/newsletter/vol1i4.html" HYPERLINK "http://nsf.gov/cise/csbytes/newsletter/vol1i4.html" HYPERLINK "http://nsf.gov/cise/csbytes/newsletter/vol1i4.html"ttissue 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

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

h HYPERLINK "http://nlp-addiction.com/chatbot" HYPERLINK "http://nlp-addiction.com/chatbot" HYPERLINK "http://nlp-addiction.com/chatbot"tVersions of the Eliza program, the first widespread chatbot.

Women in NLP Research

h HYPERLINK "http://dotdiva.org/profiles/laura.html" HYPERLINK

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

HYPERLINK "http://people.ischool.berkeley.edu/~hearst" HYPERLINKProfessor

Marti Hearst's homepage. She researches user interfaces to search engines.

```
 1 /**
 2  * A program to carry on conversations with a human
   user.
 3  * This is the initial version that:
 4  * <ul><li>
 5  *       Uses indexOf to find strings
 6  * </li><li>
 7  *          Handles responding to simple words and
   phrases
 8  * </li></ul>
 9  * This version uses a nested if to handle default
   responses.
10  * @author Laurie White
11  * @version April 2012
12  */
13 public class Magpie2
14 {
15     /**
16      * Get a default greeting
17      * @return a greeting
18      */
19     public String getGreeting()
20     {
21         return "Hello, let's talk.";
22     }
23 
24     /**
25      * Gives a response to a user statement
26      *
27      * @param statement
28      *            the user statement
29      * @return a response based on the rules given
30      */
31     public String getResponse(String statement)
32     {
33         String response = "";
34         if (statement.trim().length()==0){
35             response = "Say somethign, please.";
36         }
37         else if (statement.indexOf("no") >= 0)
38         {
39             response = "nu";
40         }
41         else if (statement.indexOf("mother") >= 0
```

```
42                  || statement.indexOf("father") >= 0
43                  || statement.indexOf("sister") >= 0
44                  || statement.contains("brother"))
45          {
46              response = "are you winning son";
47          }
48          else if(statement.contains("dog") ||
   statement.contains("cat")){
49              response = "Tell me more about your pets"
   ;
50          }
51          else if(statement.contains("Mr.Holcomb")){
52              response = "Mr.Holcomb != bad teacher";
53          }
54          else
55          {
56              response = getRandomResponse();
57          }
58          return response;
59      }
60
61      private int findKeyword(String statement, String
   goal,
62                              int startPos)
63      {
64          String phrase = statement.trim().toLowerCase
   ();
65          goal = goal.toLowerCase();
66
67          // The only change to incorporate the
   startPos is in
68          // the line below
69          int psn = phrase.indexOf(goal, startPos);
70
71          // Refinement--make sure the goal isn't part
   of a
72          // word
73          while (psn >= 0)
74          {
75              // Find the string of length 1 before and
    after
76              // the word
77              String before = " ", after = " ";
78              if (psn > 0)
```

```
 79              {
 80                  before = phrase.substring(psn - 1,
    psn);
 81              }
 82              if (psn + goal.length() < phrase.length
    ())
 83              {
 84                  after = phrase.substring(
 85                          psn + goal.length(),
 86                          psn + goal.length() + 1);
 87              }
 88
 89              // If before and after aren't letters,
    we've
 90              // found the word
 91              if (((before.compareTo("a") < 0) || (
    before
 92                      .compareTo("z") > 0)) // before
    is not a
 93                      // letter
 94                      && ((after.compareTo("a") < 0
    ) || (after
 95                      .compareTo("z") > 0)))
 96              {
 97                  return psn;
 98              }
 99
100              // The last position didn't work, so let
    's find
101              // the next, if there is one.
102              psn = phrase.indexOf(goal, psn + 1);
103
104          }
105
106          return -1;
107      }
108
109      /**
110       * Pick a default response to use if nothing
    else fits.
111       * @return a non-committal string
112       */
113      private String getRandomResponse()
114      {
```

```
115         final int NUMBER_OF_RESPONSES = 4;
116         double r = Math.random();
117         int whichResponse = (int)(r *
    NUMBER_OF_RESPONSES);
118         String response = "";
119 
120         if (whichResponse == 0)
121         {
122             response = "Interesting, tell me more.";
123         }
124         else if (whichResponse == 1)
125         {
126             response = "Hmmm.";
127         }
128         else if (whichResponse == 2)
129         {
130             response = "Do you really think so?";
131         }
132         else if (whichResponse == 4)
133         {
134             response = "ym.";
135         }
136 
137         else if (whichResponse == 5)
138         {
139             response = "dn.";
140         }
141         return response;
142     }
143 }
144 
```

```
 1 /**
 2  * A program to allow students to try out different
 3  * String methods.
 4  * @author Laurie White
 5  * @version April 2012
 6  */
 7 public class StringExplorer
 8 {
 9
10     public static void main(String[] args)
11     {
12         String sample = "The quick brown fox jumped
   over the lazy dog.";
13
14         //  Demonstrate the indexOf method.
15         int position = sample.indexOf("quick");
16         System.out.println ("sample.indexOf(\"quick\"
   ) = " + position);
17
18         //  Demonstrate the toLowerCase method.
19         String lowerCase = sample.toLowerCase();
20         System.out.println ("sample.toLowerCase() = "
    + lowerCase);
21         System.out.println ("After toLowerCase(),
   sample = " + sample);
22
23         //  Try other methods here:
24         int notFoundPsn = sample.indexOf("slow");
25
26         System.out.println("sample.indexOf(\"slow\"
   ) = " + notFoundPsn);
27         int twoParam = sample.indexOf("dog", 1);
28         System.out.println(twoParam);
29     }
30
31 }
32
```

```java
/**
 * A program to carry on conversations with a human user.
 * This version:
 *<ul><li>
 *      Uses advanced search for keywords
 *</li><li>
 *      Will transform statements as well as react to keywords
 *</li></ul>
 * @author Laurie White
 * @version April 2012
 *
 */
public class Magpie4
{
    /**
     * 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.length() == 0)
        {
            response = "Say something, please.";
        }

        else if (findKeyword(statement, "no") >= 0)
        {
            response = "Why so negative?";
        }
```

```
43          else if (findKeyword(statement, "mother") >=
    0
44                  || findKeyword(statement, "father"
    ) >= 0
45                  || findKeyword(statement, "sister"
    ) >= 0
46                  || findKeyword(statement, "brother"
    ) >= 0)
47          {
48              response = "Tell me more about your
    family.";
49          }
50
51          // Responses which require transformations
52          else if (findKeyword(statement, "I want", 0
    ) >= 0)
53          {
54              response = transformIWantToStatement(
    statement);
55          }
56
57          else
58          {
59              // Look for a two word (you <something>
    me)
60              // pattern
61              int psn = findKeyword(statement, "you", 0
    );
62
63              if (psn >= 0
64                      && findKeyword(statement, "me",
    psn) >= 0)
65              {
66                  response = transformYouMeStatement(
    statement);
67              }
68              else
69              {
70                  response = getRandomResponse();
71              }
72          }
73          return response;
74      }
75
```

```
 76     /**
 77      * Take a statement with "I want to <something>
    ." and transform it into
 78      * "What would it mean to <something>?"
 79      * @param statement the user statement, assumed
    to contain "I want to"
 80      * @return the transformed statement
 81      */
 82     private String transformIWantToStatement(String
    statement)
 83     {
 84         //  Remove the final period, if there is one
 85         statement = statement.trim();
 86         String lastChar = statement.substring(
    statement
 87                 .length() - 1);
 88         if (lastChar.equals("."))
 89         {
 90             statement = statement.substring(0,
    statement
 91                     .length() - 1);
 92         }
 93         int psn = findKeyword (statement, "I want to
    ", 0);
 94         String restOfStatement = statement.substring
    (psn + 9).trim();
 95         return "Would you really be happy if you had
     " + restOfStatement + "?";
 96     }
 97
 98
 99
100     /**
101      * Take a statement with "you <something> me"
    and transform it into
102      * "What makes you think that I <something> you
    ?"
103      * @param statement the user statement, assumed
    to contain "you" followed by "me"
104      * @return the transformed statement
105      */
106     private String transformYouMeStatement(String
    statement)
107     {
```

```
108         //  Remove the final period, if there is one
109         statement = statement.trim();
110         String lastChar = statement.substring(
    statement
111                 .length() - 1);
112         if (lastChar.equals("."))
113         {
114             statement = statement.substring(0,
    statement
115                     .length() - 1);
116         }
117 
118         int psnOfYou = findKeyword (statement, "you"
    , 0);
119         int psnOfMe = findKeyword (statement, "me",
    psnOfYou + 3);
120 
121         String restOfStatement = statement.substring
    (psnOfYou + 3, psnOfMe).trim();
122         return "Why do you " + restOfStatement + "
    me?";
123     }
124 
125 
126 
127 
128 
129     /**
130      * Search for one word in phrase. The search is
    not case
131      * sensitive. This method will check that the
    given goal
132      * is not a substring of a longer string (so,
    for
133      * example, "I know" does not contain "no").
134      *
135      * @param statement
136      *            the string to search
137      * @param goal
138      *            the string to search for
139      * @param startPos
140      *            the character of the string to
    begin the
141      *            search at
```

```
142      * @return the index of the first occurrence of
    goal in
143      *         statement or -1 if it's not found
144      */
145     private int findKeyword(String statement, String
     goal,
146             int startPos)
147     {
148         String phrase = statement.trim().toLowerCase
    ();
149         goal = goal.toLowerCase();
150 
151         // The only change to incorporate the
    startPos is in
152         // the line below
153         int psn = phrase.indexOf(goal, startPos);
154 
155         // Refinement--make sure the goal isn't part
     of a
156         // word
157         while (psn >= 0)
158         {
159             // Find the string of length 1 before
    and after
160             // the word
161             String before = " ", after = " ";
162             if (psn > 0)
163             {
164                 before = phrase.substring(psn - 1,
    psn);
165             }
166             if (psn + goal.length() < phrase.length
    ())
167             {
168                 after = phrase.substring(
169                         psn + goal.length(),
170                         psn + goal.length() + 1);
171             }
172 
173             // If before and after aren't letters,
    we've
174             // found the word
175             if (((before.compareTo("a") < 0) || (
    before
```

```
176                         .compareTo("z") > 0)) // before
    is not a
177                                                 //
    letter
178                         && ((after.compareTo("a") < 0
    ) || (after
179                                 .compareTo("z") > 0)))
180             {
181                 return psn;
182             }
183 
184             // The last position didn't work, so let
    's find
185             // the next, if there is one.
186             psn = phrase.indexOf(goal, psn + 1);
187 
188         }
189 
190         return -1;
191     }
192 
193     /**
194      * Search for one word in phrase.  The search is
     not case sensitive.
195      * This method will check that the given goal is
     not a substring of a longer string
196      * (so, for example, "I know" does not contain "
    no").  The search begins at the beginning of the
    string.
197      * @param statement the string to search
198      * @param goal the string to search for
199      * @return the index of the first occurrence of
    goal in statement or -1 if it's not found
200      */
201     private int findKeyword(String statement, String
     goal)
202     {
203         return findKeyword (statement, goal, 0);
204     }
205 
206 
207 
208     /**
209      * Pick a default response to use if nothing
```

```
209 else fits.
210      * @return a non-committal string
211      */
212     private String getRandomResponse()
213     {
214         final int NUMBER_OF_RESPONSES = 4;
215         double r = Math.random();
216         int whichResponse = (int)(r *
    NUMBER_OF_RESPONSES);
217         String response = "";
218 
219         if (whichResponse == 0)
220         {
221             response = "Interesting, tell me more.";
222         }
223         else if (whichResponse == 1)
224         {
225             response = "Hmmm.";
226         }
227         else if (whichResponse == 2)
228         {
229             response = "Do you really think so?";
230         }
231         else if (whichResponse == 3)
232         {
233             response = "You don't say.";
234         }
235 
236         return response;
237     }
238 
239 }
240 
```

```
 1 import java.util.Random;
 2
 3 /**
 4  * A program to carry on conversations with a human
   user.
 5  * This version:
 6  *<ul><li>
 7  *      Uses advanced search for keywords
 8  *</li><li>
 9  *      Will transform statements as well as react to
   keywords
10  *</li></ul>
11  * This version uses an array to hold the default
   responses.
12  * @author Laurie White
13  * @version April 2012
14  */
15 public class Magpie5
16 {
17     public static int run = 0;
18     /**
19      * Get a default greeting
20      * @return a greeting
21      */
22     public String getGreeting()
23     {
24         return "Hello, let's talk.";
25     }
26
27     /**
28      * Gives a response to a user statement
29      *
30      * @param statement
31      *            the user statement
32      * @return a response based on the rules given
33      */
34     public String getResponse(String statement)
35     {
36         String response = "";
37         if (statement.length() == 0)
38         {
39             response = "Say something, please.";
40         }
41
```

```
42          else if (findKeyword(statement, "no") >= 0)
43          {
44              response = "Why so negative?";
45          }
46          else if (findKeyword(statement, "mother") >=
    0
47                  || findKeyword(statement, "father"
    ) >= 0
48                  || findKeyword(statement, "sister"
    ) >= 0
49                  || findKeyword(statement, "brother"
    ) >= 0)
50          {
51              response = "Tell me more about your
    family.";
52          }
53
54          // Responses which require transformations
55          else if (findKeyword(statement, "I want to",
    0) >= 0)
56          {
57              response = transformIWantToStatement(
    statement);
58          }
59          //  Part of student solution
60          else if (findKeyword(statement, "I want", 0
    ) >= 0)
61          {
62              response = transformIWantStatement(
    statement);
63          }
64
65          else
66          {
67
68              // Look for a two word (you <something>
    me)
69              // pattern
70              int psn = findKeyword(statement, "you", 0
    );
71
72              if (psn >= 0
73                      && findKeyword(statement, "me",
    psn) >= 0)
```

```
74              {
75                  response = transformYouMeStatement(
   statement);
76              }
77              else
78              {
79                  //  Part of student solution
80                  // Look for a two word (I <something
   > you)
81                  // pattern
82                  psn = findKeyword(statement, "i", 0
   );
83
84                  if (psn >= 0
85                          && findKeyword(statement, "
   you", psn) >= 0)
86                  {
87                      response =
   transformIYouStatement(statement);
88                  }
89                  else
90                  {
91                      response = getRandomResponse();
92                  }
93              }
94          }
95          return response;
96      }
97
98      /**
99       * Take a statement with "I want to <something>
   ." and transform it into
100      * "What would it mean to <something>?"
101      * @param statement the user statement, assumed
   to contain "I want to"
102      * @return the transformed statement
103      */
104     private String transformIWantToStatement(String
   statement)
105     {
106         //  Remove the final period, if there is one
107         statement = statement.trim();
108         String lastChar = statement.substring(
   statement
```

```
109                 .length() - 1);
110         if (lastChar.equals("."))
111         {
112             statement = statement.substring(0,
    statement
113                     .length() - 1);
114         }
115         int psn = findKeyword (statement, "I want to
    ", 0);
116         String restOfStatement = statement.substring
    (psn + 9).trim();
117         return "What would it mean to " +
    restOfStatement + "?";
118     }
119 
120 
121     /**
122      * Take a statement with "I want <something>."
    and transform it into
123      * "Would you really be happy if you had <
    something>?"
124      * @param statement the user statement, assumed
    to contain "I want"
125      * @return the transformed statement
126      */
127     private String transformIWantStatement(String
    statement)
128     {
129         //  Remove the final period, if there is one
130         statement = statement.trim();
131         String lastChar = statement.substring(
    statement
132                 .length() - 1);
133         if (lastChar.equals("."))
134         {
135             statement = statement.substring(0,
    statement
136                     .length() - 1);
137         }
138         int psn = findKeyword (statement, "I want",
    0);
139         String restOfStatement = statement.substring
    (psn + 6).trim();
140         return "Would you really be happy if you had
```

```
140  " + restOfStatement + "?";
141     }
142
143     /**
144      * Take a statement with "you <something> me"
    and transform it into
145      * "What makes you think that I <something> you
    ?"
146      * @param statement the user statement, assumed
    to contain "you" followed by "me"
147      * @return the transformed statement
148      */
149     private String transformYouMeStatement(String
    statement)
150     {
151         //  Remove the final period, if there is one
152         statement = statement.trim();
153         String lastChar = statement.substring(
    statement
154                 .length() - 1);
155         if (lastChar.equals("."))
156         {
157             statement = statement.substring(0,
    statement
158                     .length() - 1);
159         }
160
161         int psnOfYou = findKeyword (statement, "you"
    , 0);
162         int psnOfMe = findKeyword (statement, "me",
    psnOfYou + 3);
163
164         String restOfStatement = statement.substring
    (psnOfYou + 3, psnOfMe).trim();
165         return "What makes you think that I " +
    restOfStatement + " you?";
166     }
167
168     /**
169      * Take a statement with "I <something> you" and
     transform it into
170      * "Why do you <something> me?"
171      * @param statement the user statement, assumed
    to contain "I" followed by "you"
```

```
172      * @return the transformed statement
173      */
174     private String transformIYouStatement(String
    statement)
175     {
176         //  Remove the final period, if there is one
177         statement = statement.trim();
178         String lastChar = statement.substring(
    statement
179                 .length() - 1);
180         if (lastChar.equals("."))
181         {
182             statement = statement.substring(0,
    statement
183                     .length() - 1);
184         }
185 
186         int psnOfI = findKeyword (statement, "I", 0
    );
187         int psnOfYou = findKeyword (statement, "you"
    , psnOfI);
188 
189         String restOfStatement = statement.substring
    (psnOfI + 1, psnOfYou).trim();
190         return "Why do you " + restOfStatement + "
    me?";
191     }
192 
193 
194 
195 
196     /**
197      * Search for one word in phrase. The search is
    not case
198      * sensitive. This method will check that the
    given goal
199      * is not a substring of a longer string (so,
    for
200      * example, "I know" does not contain "no").
201      *
202      * @param statement
203      *            the string to search
204      * @param goal
205      *            the string to search for
```

```
206      * @param startPos
207      *            the character of the string to
    begin the
208      *            search at
209      * @return the index of the first occurrence of
    goal in
210      *         statement or -1 if it's not found
211      */
212     private int findKeyword(String statement, String
     goal,
213             int startPos)
214     {
215         String phrase = statement.trim().toLowerCase
    ();
216         goal = goal.toLowerCase();
217 
218         // The only change to incorporate the
    startPos is in
219         // the line below
220         int psn = phrase.indexOf(goal, startPos);
221 
222         // Refinement--make sure the goal isn't part
     of a
223         // word
224         while (psn >= 0)
225         {
226             // Find the string of length 1 before
    and after
227             // the word
228             String before = " ", after = " ";
229             if (psn > 0)
230             {
231                 before = phrase.substring(psn - 1,
    psn);
232             }
233             if (psn + goal.length() < phrase.length
    ())
234             {
235                 after = phrase.substring(
236                         psn + goal.length(),
237                         psn + goal.length() + 1);
238             }
239 
240             // If before and after aren't letters,
```

```
240 we've
241             // found the word
242             if (((before.compareTo("a") < 0) || (
    before
243                     .compareTo("z") > 0)) // before
    is not a
244                                           //
    letter
245                     && ((after.compareTo("a") < 0
    ) || (after
246                             .compareTo("z") > 0)))
247             {
248                 return psn;
249             }
250
251             // The last position didn't work, so let
    's find
252             // the next, if there is one.
253             psn = phrase.indexOf(goal, psn + 1);
254
255         }
256
257         return -1;
258     }
259
260     /**
261      * Search for one word in phrase.  The search is
     not case sensitive.
262      * This method will check that the given goal is
     not a substring of a longer string
263      * (so, for example, "I know" does not contain "
    no").  The search begins at the beginning of the
    string.
264      * @param statement the string to search
265      * @param goal the string to search for
266      * @return the index of the first occurrence of
    goal in statement or -1 if it's not found
267      */
268     private int findKeyword(String statement, String
     goal)
269     {
270         return findKeyword (statement, goal, 0);
271     }
272
```

```
273
274
275     /**
276      * Pick a default response to use if nothing
    else fits.
277      * @return a non-committal string
278      */
279     private String getRandomResponse ()
280     {
281         if (run == 0){
282             run++;
283             return randomResponses[7];
284         }
285         Random r = new Random ();
286         return randomResponses [r.nextInt(
    randomResponses.length)];
287     }
288
289     private String [] randomResponses = {"
    Interesting, tell me more",
290             "Hmmm.",
291             "Do you really think so?",
292             "You don't say.",
293             "ym",
294             "1773",
295             "dn",
296             "Exception in thread \"main\" java.lang.
    StackOverflowError\n" +
297                     "        at sun.nio.cs.
    StreamEncoder.writeBytes(StreamEncoder.java:221)\n"
     +
298                     "        at sun.nio.cs.
    StreamEncoder.implFlushBuffer(StreamEncoder.java:291
    )\n" +
299                     "        at roses.are.red.
    violets(are.blue:,missing)curlybrace\n" +
300                     "        on line.32.PrintStream.
    write(PrintStream.java:480)\n" +
301                     "        at sun.nio.cs.
    StreamEncoder.flushBuffer(StreamEncoder.java:104)\n"
     +
302                     "        at java.io.
    OutputStreamWriter.flushBuffer(OutputStreamWriter.
    java:185)\n" +
```

```
303                         "        at java.io.PrintStream.
    print(PrintStream.java:669)\n" +
304                         "        at java.io.PrintStream.
    println(PrintStream.java:806)\n" +
305                         "        at
    StackOverflowErrorExample.recursivePrint(
    StackOverflowErrorExample.java:4)\n" +
306                         "        at
    StackOverflowErrorExample.recursivePrint(
    StackOverflowErrorExample.java:9)\n" +
307                         "        at
    StackOverflowErrorExample.recursivePrint(
    StackOverflowErrorExample.java:9)\n" +
308                         "        at
    StackOverflowErrorExample.recursivePrint(
    DonotpressenterDonotcollect100dollers.java:9)"
309     };
310 }
311
```