**Advanced Placement Computer Science Programming Assignment 9.0  
Assigned March 2, 2015 Due: March 6, 2015**

**Derek Wider**

**More Activities with the Magpie Class**

**Activity 1: Better Keyword Detection**

In the previous Magpie activities, 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**

You should still have available the following, if not I will post them on the website on the assigned date:

1. the API for the Magpie classr2
2. the API for the String class
3. the code for the StringExplorer
4. the code for the Magpie
5. the code for the MagpieRunner

**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();

int psn = phrase.toLowerCase().indexOf(goal.toLowerCase(), startPos);

while (psn >= 0)

{

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();

}

**/\* 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.toLowerCase(), 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 |
|  | 6 | ”r” | “ “ |
| 2 | 15 | “o” | “ ‘ “ |
| 3 | 21 | “ “ | “ “ |

Use a copy of the table below to trace the calls (Doing the “no” call)

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | psn | Before | after |
|  | 3 | “k” | “w” |
| 2 | 7 | “ “ | “t” |
| 3 | 21 | “s” | “w” |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Exercise: Use the new method**

Repeat the changes you made to the program in the previous assignment, using this new method to detect keywords.

(see magpie3)

**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 2: 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.

**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?

* 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?

This structure (I something you) does not work when the verb is in the past tense (ex. I liked you). This can be fixed by reading in the last (non-space) character and seeing if it is a “d”, indicating past tense.

**Activity 3: Arrays and the Magpie**

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.

**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.