AP® Computer Science A Magpie Chatbot Lab Teacher's Guide

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

Magpie Chatbot Teacher's Guide

Overview

From Eliza in the 1960s to Siri and Watson today, the idea of talking to computers in natural language has fascinated people. (And yes, there have been more than one conversation between Eliza and Siri – search for "chatbot Eliza Siri."[1]) While Natural Language Processing (NLP) is a complicated field, it is fairly easy to create a simple program to respond to English sentences.

This lab will take strings, parse them for recognizable information, and respond according to the information found. This can be done early in the course, as soon as students have learned `if` statements and start working with the `String` class. The code contains a `while` loop, but students only need to be able to read this code instead of modify it, so the lab can be used to introduce these concepts. There are suggested extensions which can be used to introduce arrays and array lists later in the class.

Learning Objectives

This lab is suitable for use early in the course. The primary topics covered in this lab relate to `if` statements and algorithms, using the `String` class to search for key phrases and restructure the user statement (topics II. B. 4 and II. C in the AP Computer Science A Course Description outline). Additionally, students will do extensive modification of existing code (topic III. C.).

Prerequisites

Before doing this lab, students should be able to:

- locate WWW sites using a browser
- run and edit Java programs that use classes
- trace code
- read APIs for both the standard Java libraries and local code
    - If desired, APIs can be introduced with the second activity. Extra time should be allocated to allow students to read them and see other examples.

There are two additional activities that incorporate arrays and `ArrayLists` into the Magpie lab. If the Magpie lab is successful with your class, you can include these activities when you cover arrays and `ArrayLists`.

[1] Rather than provide links to specific WWW sites that may change, this lab will provide search terms which are likely to provide useful sites. While the terms will be given in quotes, they should not be quoted when used in a search.

What Is Provided

The code for the Magpie in its progressive forms is provided.

Installation/Setup

Teachers will need to have computers with Java development tools and access to the Internet, though access to the Internet is only needed for the first activity. If a class cannot access the chatbot.org site, alternative introductions are discussed. The files from the Magpie activities will need to be copied to student development environments.

Activity Outline

Activity 1 (Optional)

Students explore existing chatbots and try them on predefined statements. Ideally, students will work in small groups as they try out the chatbots. Working in pairs usually encourages more exploration. They can present their work at the end of the class if more time is available.

The chatbots.org site has a lot of information. Students who are good at finding data on WWW sites should find plenty here, but those who are a bit overwhelmed can be directed to the Awards tab. There is a list of award-winning chatbots available on this page (it is currently in the right-hand column). Selecting one of these will bring the student to a page for the chatbot. From there, the student can open the chatbot on the WWW (this choice is currently in the middle column, under the Chat Now title).

Alternatively, teachers can provide links to specific chatbots, such as http://www.elbot.com and http://www.pandorabots.com/pandora/talk?botid=f5d922d97e345aa1.

There is a much shorter list of chatbots at http://nlp-addiction.com/chatbot, which is easier to navigate. Searching for "chatbot Eliza" gives numerous sites that have chatbots or links to chatbots. There are also some YouTube videos of the Loebner competition, accessible from the Awards tab, that can be used to demonstrate chatbots if students cannot access the Internet directly. (These are unlisted and not accessible through a regular search.)

Because some schools restrict access to the Internet, students may not be able to perform this lab. Instead, teachers can download a chatbot or show videos of conversations with different chatbots.

This activity can expand to fit 15 to 45 minutes.

Activity 2

Students work with the Magpie code for the first time. They are guided through exercises and given the chance to add their own keyword/response pairs.

The students will work with the code in the `Magpie2.java` file. A simple runner is available in `MagpieRunner2.java`.

This activity will take from one to two hours. Students should work in small groups, which will help them in developing their own keyword/response pairs.

Activity 3

Students are first exposed to the Java API. The link to this has changed from time to time, but a search for Java API usually gives it as the first result. If there is no Internet access for students, it can be downloaded (currently from http://www.oracle.com/technetwork/java/javase/downloads/index.html, search for "Java API") and installed locally. The current API should match the current version of Java supported by the exam (at this writing, that version is Java 6, but Java 7 has been released and may be the standard, and Java 8 is on the horizon). This portion of the activity may easily be separated from the second part, and used even earlier in the course.

There is a lot of material in the documentation, but a benefit of Java is that the documentation is extracted directly from the source code. When the code is changed, it is easy to change the documentation at the same time. Comments that will be used for documentation are in a style known as Javadoc. Javadoc comments begin with `/**` instead of `/*`. While Javadoc is not testable, code on the test often uses Javadoc comments, so it is beneficial if students have experience reading Javadoc-style comments. Because Javadoc is used to produce HTML, there are some HTML commands in the Javadoc to help format the code. These can be ignored as you read the comments in the programs.

Students continue the activity by tracing a more complex method to find keywords in a string. In doing this, they will be exposed to more advanced `String` methods. The API should be useful in understanding how these work. This may be the first introduction to the `while` loop for some students. They then redo Activity 2 using this new method.

This activity will take from one to two hours, depending on how adept students are at reading code, their previous exposure to the API, and their knowledge of `String` methods.

Activity 4

This final required activity has students working with responses that are transformations of the original statement. It can be made as simple or complex as teachers want. As is, it will take one to two hours to complete.

Extension Activity 5

The Magpie lab is designed to be used early in the class, but it can be useful later in the class as well. This fairly short activity provides a rewrite of the `getRandomResponse` method to use an array. It can be used to introduce arrays or early in the discussion of arrays. Students add literal strings to an array of random responses; this illustrates use of the length attribute of arrays. It should take no more than 30 minutes to complete.

Answers to Questions and Exercises

Solutions to coding questions are in the corresponding "complete" folder. For example, the solution to Activity 1 is in `activity1complete`.

Group Work

All of these activities can be done in pairs. Students tend to be more adventurous with someone beside them and will often try out more things in the first activity. They can compare their custom keywords and responses in other activities.

Assessment

Multiple-Choice Questions

No questions will come directly from this material and it will not be provided as part of the Quick Reference for the exam. Instead, this material will lead to two types of multiple-choice questions: complex conditional questions and string manipulation questions.

1. Consider the following code segment.

```
if (a < b)
{
  if (b < c)
  {
    if (c < 10)
    {
      System.out.println("one")
    }
    else if (c < a)
```

```
    {
      System.out.println("two")
    }
  }
}
else
{
  if (c < a)
  {
    System.out.println("three")
  }
  else
  {
    System.out.println("four")
  }
}
```

For which values of `a`, `b`, and `c` will the code print "one"?

I. `a = 5, b = 6, c = 7`

II. `a = 8, b = 7, c = 6`

III. `a = 10, b = 20, c = 30`

(A) I only

(B) II only

(C) III only

(D) I and III

(E) I and II

2. What is the output of the following code segment?

```
String phrase = "Here is the word";
int psn = phrase.indexOf("e");
while (psn >= 0)
{
   System.out.print(psn + " ");
   phrase = phrase.substring(psn + 1);
   psn = phrase.indexOf("e");
}
```

(A) `1 1 6`

(B) `2 2 7`

(C) `1 3 10`

(D) `2 4 11`

(E) Many digits will be printed due to an infinite loop.

Answers to Multiple-Choice Questions

1. (A) For more practice, teachers can also ask when "`two`," "`three`," or "`four`" would be printed.

2. (A)

Free-Response Question

Free-response questions related to this lab will typically focus on text processing. One possible question is shown below.

When WWW developers want to convert plain text documents to HTML, they need to make changes to the text to insert tags and replace certain characters. For this question, you will write part of the class to convert text to HTML. You will write methods to replace underscores (`"_"`) with tags indicating the text should be in italics.

The class definition for this problem is given below:

```
public class TextFormatter
{
  private String line; //  The line to format

  public TextFormatter (String lineToFormat)
  {  line = lineToFormat;  }


  /**
   * Finds the first single instance of str in line,
   * starting at the position start
   * @param str the string of length 1 to find.
   * Guaranteed to be length 1.
   * @param start the position to start searching.
   * Guaranteed to be in the string line.
   * @return the index of the first single instance of
   * str if the string is found or -1 if it is not found.
   */
  private int findString (String str, int start)
  {  /* To be implemented in part a) */ }
```

```
  /**
   * Count the number of times single instances of str appear in
   * the line.
   * @param str the string to find.
   * Guaranteed to be length 1.
   * @return the number of times the string appears in the line
   */
  private int countStrings (String str)
  {  /* To be implemented in part b) */ }


  /**
   * Replace all single instances of underscores in the line given by
   * line with italics tags.  There must be an even number of underscores
   * in the line, and they will be replaced by <I>, </I>, alternating.
   * @param original a string of length 1 to replace
   * @param replacement the string (of any length) use as a replacement
   * @return the line with single instances of underscores replaced with
   * <I> tags or the original line if there are not an even number of
   * underscores.
   */
  public String convertItalics ()
  {  /* To be implemented in part c) */ }
}
```


a) Write the method `findString`. This method will take a string of length 1 to find in the line, at a given starting point. It will return the location of the goal string. This differs from the `indexOf` method of the `String` class because it requires the string be just a single instance of the string, so if the string appears two or more times consecutively, it will not return any of those values. If there is no single instance, the method should return -1. Consider the following examples, where `line` has the value `"aabaccb"`.

Call	Result	Explanation
`findString("a", 0)`	3	The first single occurrence of `"a"` is in position 3. The `"a"s` in position 0 and 1 are not returned because they are not single instances.
`findString("b", 4)`	6	The first single occurrence of `"b"` at or after position 4 is in position 6.
`findString("c", 0)`	-1	There is no single instance of `"c"` in the line.

b) Write the method `countStrings`. This method will take a string of length 1 to find in the line. It will return the number of times single instances of the goal string appear in the string.

c) Write the method `convertItalics`. If the line has an even number of single underscores, a line with those underscores converted to alternating <I> and </I> tags should be returned. The first underscore will be converted to <I>, the second to </I>, the third to <I>, and so on. If the line does not have an even number of <I> tags, the original line should be returned. Consider the following examples.

`line`	value returned by `convertItalics`
`This is _very_ good.`	`This is <I>very</I> good.`
`_This_ is _very_ _good_.`	`<I>This</I> is <I>very</I> <I>good</I>.`
`This is _very good.`	`This is _very good.`
`This is __very good.`	`This is __very good.`

Possible Solution to the Free-Response Question

The complete `TextFormatter` class is below. Teachers who want to go further with this example, as a homework assignment or class example, can add handling asterisks for the <B> tag and/or substituting for special characters (for example, `"<"` is used instead of a less than sign—`"<"`). While just adding a second tag like an asterisk seems to be simple, extra care will need to be taken to ensure that tags are nested correctly. For example, the line `"Here *is a _bad* example_"` has invalid nesting.


```
public class TextFormatter
{
  private String line; //  The line to format

  public TextFormatter (String lineToFormat)
  {  line = lineToFormat;  }


  /**
   * Finds the first single instance of str in line,
   * starting at the position start
   * @param str the string of length 1 to find.
   * Guaranteed to be length 1.
   * @param start the position to start searching.
   * Guaranteed to be in the string line.
   * @return the index of the first single instance of
   * str if the string is found or -1 if it is not found.
   */
  private int findString (String str, int start)
```

```
{
  int psn = line.indexOf(str, start);
  while (psn >= 0) {
    //  find the string of length 1 before and after the instance found
    String before = "";
    String after = "";
    if (psn > 0)
      before = line.substring(psn - 1, psn);
    if (psn < line.length() - 1)
      after = line.substring(psn + 1, psn + 2);

    //  If the before and after are different from
    //  this string, we've found the position.
    if (!before.equals(str) && !after.equals(str))
      return psn;

    //  If not, look for the next position
    psn = line.indexOf (str, psn + 1);
  }
  //  Otherwise, the string isn't found.
  return -1;
}

/**
 * Count the number of times single instances of str appear in
 * the line.
 * @param str the string to find.
 * Guaranteed to be length 1.
 * @return the number of times the string appears in the line
 */
private int countStrings (String str)
{
  int count = 0;
  int psn = 0;
  while (findString (str, psn) >= 0)
  {
    count++;
    psn = findString (str, psn) +1;
  }
  return count;
}

/**
 * Replace all single instances of underscores in the line given by
 * line with italics tags.  There must be an even number of underscores
 * in the line, and they will be replaced by <I>, </I>, alternating.
 * @param original a string of length 1 to replace
 * @param replacement the string (of any length) use as a replacement
 * @return the line with single instances of underscores replaced with
```

```
   * <I> tags or the original line if there are not an even number of
   * underscores.
   */
  public String convertItalics ()
  {
    //  Ensure there are an even number of underscores.
    if (countStrings ("_") % 2 == 1)
      return line;

    String tag = "<I>";
    String result = "";
    int psn = 0;

    while (findString ("_", psn) >= 0 )
    {
      int newPsn = findString ("_", psn);
      //  Copy the code before the "_" into the result
      result = result + line.substring(psn, newPsn);

      //  Add the tag and change the tag
      result = result + tag;
      if (tag.equals ("<I>"))
        tag = "</I>";
      else
        tag = "<I>";

      //  update the psn
      psn = newPsn + 1;
    }
    //  copy the rest of the string
    result = result + line.substring(psn);

    return result;
  }

}
```

Extensions

Instance Variables

To show the use of instance variables, a constructor can be added that takes the name of the user. The Magpie can then use the name occasionally in responses.

Adding Domain Knowledge

With more knowledge of the domain, more interesting responses can be made. One possible programming assignment is to have students pick a domain (like Northeast's student government or Central High School football) and create a Magpie that can speak with confidence about it. For example, if the student government president's name is used, the Magpie can recognize that the person is the president and it can respond to specific school issues, like recognizing the theme of the prom.

Alternatively, if a more structured input format is required, much more domain knowledge can be used. Consider a program that can respond to geometric questions like:

What is the area of a circle with diameter 7?

What is the circumference of a square with side 3?

Alternatively, the program can respond to questions about state capitals, help with language translation, etc.

Incorporating Array Lists

Activities planned for later in the course can be based on the ground work set here. For example, once array lists are introduced, the Magpie can be revisited and given a memory of the statements and responses. Instead of picking a default response, the Magpie can return to a previous topic if there's nothing it can respond to in the current statement.

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.