Term 2 – Week 3 Name:

**Magpie Lab: Part 1 – Activity 1 – Overview**

**Overview – What is a ChatBot?**

A ChatBot is a program designed to simulate conversation and is a part of the branch of computer science known as Artificial Intelligence. The main emphasis of this lab is String processing similar to what we have seen in the Tweet Tester and Wild Card labs. Today you will try out some ChatBots to see what they do.

Link: http://www.pandorabots.com/pandora/talk?botid=f5d922d97e345aa1

1. Open the link above. Try out several of the ChatBots linked there like Fake Captain Kirk or Claudio.

2. Pick two to compare. How do they respond to the following:

**ChatBot 1 ChatBot 2**

a. Hi there A big hello to you, as well. Hello! Have you a question for me?

b. I like apples So you are into different kinds of fruit! What does that feel like? Apples are not good conversationalists.

c. Where are you You tell me where I am. It's confusing in here. I am in the computer at Pandorabots.com.

d. How are you I feel terrible today. I am fine, thank you.

3. Comparing these answers, how were the two ChatBots the same? Different?

Both answered my questions but in different ways

4. Many ChatBots work by searching for specific keywords, like **Where**. What keywords did your two ChatBots use?

Hi, like, apples, where, how, are, you,

5. As the field of Artificial Intelligence grows how do ChatBots change how we interact with technology? How do you think they might be used as smart phones and the Internet become more widespread?

They are being used in phones by Digital assistance, like siri, google assistant, cortana, bixby.

Magpie Lab: Part 2 – Activity 2 – First Look at the Code

To explore the code make sure you have all the Magpie lab materials saved on your computer. The Magpie lab consists of several files in different directories. Each Activity is in a separate folder in the guide.

Today we will be working through the first part of Activity 2. The files can be found here:

**MagpieActivityStarterCode\activity2.**

1. Open the files **MagpieRunner2.java** and **Magpie2.java**. Compile both and run the Runner program. What does it do?

It first says Hello, let’s talk and begins a conversation.

1. How does the Runner program stop repeating the conversation?

It doesn’t and if it is it’s doing a very bad job at doing so.

1. As you use the program, what keywords do you notice?

Hi, no, yes

4. Look at the **Magpie2.java** file. What do each of the following methods do:

1. **getGreeting()**

Returns a String saying “ Hello, let’s talk.”

1. **getResponse(String statement)**

Returns “Why so negative?” if what I typed in contained a “no”, returns “Tell me more about your family.” If what I typed contained “mother, father, sister or brother” and if it didn’t meet any of that criteria it would run getRandomResponse().

1. **getRandomResponse()**

It has a constant which is 4 and a variable that is a random number between 0 and 1 and it multiplies the constant with the variable and casts it into an Int to get either 0,1,2,3 or 4 and depending on which number it landed on it will return either, "Interesting, tell me more.", "Hmmm.", "Do you really think so?" or "You don't say.", respectively.

5. Compare the keyword list you developed in question 3 above to the if statement in the **getResponse()**method. How would you add more keywords to this ChatBot?

I would add more responses, and change the number of responses constant to the number of responses that there now are.

**Magpie Lab: Part 3 – Activity 2 – Changing the Code**

To explore the code make sure you have all the Magpie lab materials saved on your computer. The Magpie lab consists of several files in different directories. Each Activity is in a separate folder in the guide.

Today we will be working through the second part of Activity 2. The files can be found here:

**MagpieActivityStarterCode\activity2.**

1. Open the files **MagpieRunner2.java** and **Magpie2.java**.

2. Complete Activity 2 in the Magpie Student Guide on page 3.

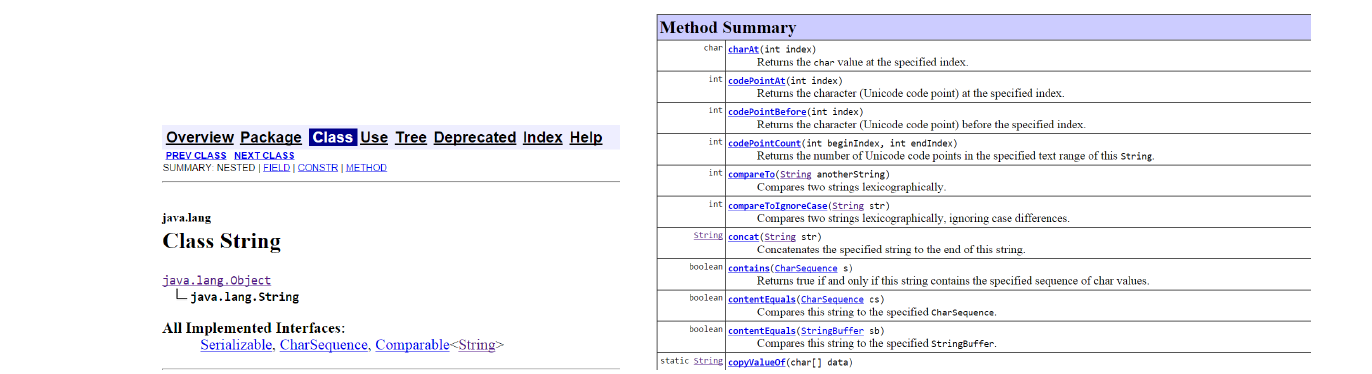
3. Answer the question on page 4 here:

The problem with those statements and the responses that the program returns is that since the way the program checks if what was typed in has something to do with Family or No, is that it checks if the string that was typed in contains the word “no” or “mother” using IndexOf but by *using if (statement.indexOf("mother") >= 0*  it doesn’t check if it is a separate word or apart of another word. If it was to check if it was a part of another word or not, it would have to check the index of “ no “ instead of “no”.

**Magpie Lab: Part 4 – Activity 3 – String APIs**

This activity asks you to work with the String class. While we covered this class back in Term 1, this activity asks you to investigate and use some of the methods in the String class that are not covered on the APCS A Exam.

In order to use these methods you need to read the String class' API. An API (Application Programming Interface) is a listing of the features and behaviors of a system that tell a programmer how to use them. We have looked at these several times this year in the lessons and webinars, specifically for the Math and String classes.



This screenshot on the left shows the String API. This information at the top shows where the String class fits in to the hierarchy of all of the Java classes, and gives some information about how it works. On the right the screenshot shows the beginning of the method summary listing everything available through the String class.

**Use the String API to answer the following:**

Link: <http://docs.oracle.com/javase/8/docs/api/java/lang/String.html>

1. How many constructors are in the String class?

There are 15 constructors in the String Class.

2. How many **indexOf** methods are in the String class?

There are 4 indexOf methods in the String Class.

3. What does the **replaceAll(String regex, String replacement)** method do?

Replaces each substring of this string that matches the given regular expression with the given replacement.

4. Are there any methods there that would have made the TweetTester lab easier to complete?

We never did that lab.

**Magpie Lab: Part 5 – Activity 3 – Magpie API**

In the last worksheet you explored the String API. These API's exist for all classes built into Java and in fact any

programmer can create an API for a new class they create.

1. Open The Magpie3 lab. Look at the **findKeyword** method. Read the comments above this method. What does the method appear to do?

It finds the keyword put into the statement parameter when it’s not connected to any other word but by itself.

1. ~~Ignore this question. JavaDoc is a specific way of coding comments so that an HTML file with the class's API can be created. The comments you see in the~~ **~~findKeyword~~** ~~method follow this format. Notice it starts with /\*\* and uses several keywords with @ in front. These tell Java how to create the HTML page with the API.~~

~~Dr. Java has a simple way of creating JavaDoc from our code. Go to the~~ **~~Tools~~** ~~menu and select~~ **~~Preview JavaDoc~~**

**~~for Current Document~~**~~. This should open up an HTML file in your browser in the JavaDoc format.~~

1. In the Magpie3 API open in your browser look for the **findKeyword** method, you will notice it is not there. Look back at the code for the **findKeyword** method. What Java keyword in the header (first line) of the method might make this not show up in the JavaDoc?

The word “private”

1. Change the **findKeyword** method from **private** to **public**. Recompile, then using the JavaDoc tool preview the API. What is different?

The findKeyword method now shows up in the JavaDoc.

1. Looking at the JavaDoc for the **findKeyword** method what do the following appear to do?

|  |  |
| --- | --- |
| a. @param statement | The string that you want to search in |

|  |  |
| --- | --- |
| b. @param goal | The string that you want to search for |

|  |  |
| --- | --- |
| c. @param startPos | Where you’d like to start the search in the string |

|  |  |
| --- | --- |
| d. @return | Where the first occurrence on the string is or -1 if it cannot be found. |

Write your comment here:

This method does nothing. @param dud Is not used for anything.

1. Let's try creating JavaDoc from scratch. Add a method to Magpie3:

**public void doNothing (int dud)**

**{ //does nothing }**

1. Why do you think Java allows programmers to create APIs using JavaDoc?

To make it easy to document and comment their code so others and even themselves to understand and interpret.

Magpie Lab: Part 6 – Activity 3 – The findKeyword Method

In this worksheet you will finish up Activity 3 by exploring what the **findKeyword** method does.

Open the **Magpie3.java** file and use it to answer the following questions.

If you changed the **findKeyword** method to be **public** in Part 5 of this guide you should change it back to **private** and recompile.

1. Since the **findKeyword** method is private it cannot be called from a runner program. Where in the Magpie3 file is the code called?

The method is called inside of the getResponce method to check if they typed the word no or anything to do with family.

1. There are two versions of the **findKeyword** method in the Magpie3 lab. What is the difference between the two headers?

Having two methods with the same name in the same class is called

Overloading

Parameters

Java tells which of the two versions to use based only on the

1. While the two versions of the **findKeyword** method performs a similar task, they do so in a slightly different way. How are they different?

One will begin the search starting at wherever you tell it to and the other will start at the beginning of the string.

1. On page seven of Activity 3 complete the chart tracing the **findKeyword** method.

findKeyword("I know nothing about snow plows and so I have no job.", "no", 0);

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | psn | before | after |
| 1 | 3 | k | w |
| 2 | 7 |  | t |
| 3 | 22 | s | w |
| 4 | 45 |  |  |

**Magpie Lab: Part 7 – Activity 4 – More responses**

This section of the Lab implements more sophisticated ChatBot algorithms. In this version some keywords, like

**want** cause the Magpie to respond using you own words.

Open the Magpie4 Lab and Runner and compile them both, then run the Runner program.

1. Try chatting. Are there any new keywords you notice?

Like and want.

1. Try the following phrases. What response do you get?

What makes you think that I like you? What would it mean to program? What makes you think that I type you?

a. I want to program.

b. Do you like cheese?

c. Do you like me?

d. You like me?

e. You type me.

What patterns do you recognize in these phrases?

They are relevant to my questions and all comeback as more questions.

3. Work through the exercises on the next two pages

**Magpie Lab: Part 8 – Activity 5 – Using Arrays**

This version of the Magpie Lab behaves in the same way as the version in Activity 4. The difference is how it

generates a random response when no keywords are found.

Open the Magpie4 Lab and Runner and compile them both, then run the Runner program.

1. Try chatting. How does the behavior compare to version 4 of Magpie:

It has more responses and they’re related to what you type in.

1. At the bottom of the Magpie5 Lab find the code:

**private String [] randomResponses = {"Interesting, tell me more",**

**"Hmmm.",**

**"Do you really think so?",**

**"You don't say."**

**};**

This array of Strings holds the responses the ChatBot randomly says. Add two more random responses and compile, then run the runner for Magpie5.

1. Look at the **getRandomResponse** method in the Magpie5 Lab and compare it to the same method in the **getRandomResponse** method in Magpie4. Compare and contrast how the two methods work.
2. Let's say you wanted to increase the number of random responses to 30 different phrases. Which of the two methods, the one in Magpie 4 or Magpie5, would be most efficient? Why?
3. How does storing the responses in array enable increasing the number of random responses the Magpie ChatBot can use?