ITSC 1212 Module 7 Lab

In this lab we will continue using the chatbot example from the previous lab.

Concepts covered in this lab:

- Compound conditionals using logical operators AND and OR
- NOT logical operator
- Strings and String methods

Required files or links

The files we'll be using are the same as those from the previous lab. If you have completed the previous lab successfully you can continue from where you left off. If you had problems with that lab you can start fresh but redownloading the files and adding them to your VSCode folder. Keep in mind that you can not have two files in the same folder with identical names, so you will have to rename them to something else (e.g., Chatbot.java and ChatbotDriver.java):

- Magpie.java
- MagpieDriver.java

Part A:

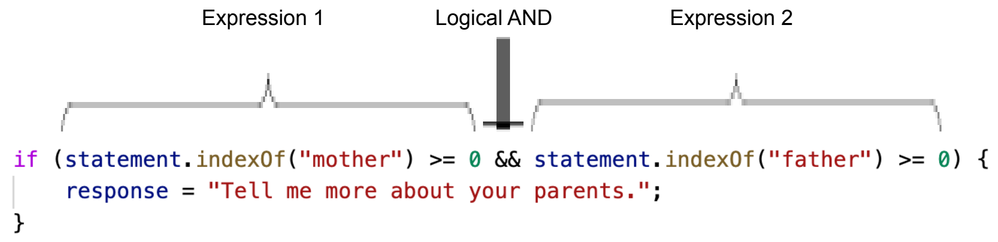
In this part you will practice using the logical AND operator (two ampersands && in Java) to connect individual conditions.

1. Refer to the previous lab to refresh your memory with the contents of the Magpie.java and MagpieDriver.java class files.
2. Open the Magpie.java file in your IDE. Recall from the previous lab the getResponse(...) method role is to check the String entered by the user (stored in the String variable **userEntry**) for certain keywords and sets the response accordingly when one of the keywords is found.
   Remember also that since this is a series of independent IF statements instead of nested If-ELSE statements, the last IF statement with a true condition will control the value of the response generated.

   In this part of the lab we want to look for situations where both of two Strings are found in the user's entry then generate a specific response. Looking for the presence of two Strings can be achieved by using a compound conditional. That is, two or more conditional expressions, in this case joined by the logical operator AND. For example, if we wanted the chatbot to respond with "Tell me more about your parents" when the statement includes both the word "mother" **and** the word "father", we can structure the IF statement like this:

```
if (statement.indexOf("mother") >= 0 && statement.indexOf("father") >= 0) {
    response = "Tell me more about your parents.";
}
```

How does this work? There are two conditional expressions: Expression 1 determines if the String “mother” is present and Expression 2 determines if the String “father” is present. Each of these expressions will be either true or false. The two expressions are joined by the logical AND operator (&&) to determine if the compound expression (everything between the parentheses) is true or false:



```java
if (statement.indexOf("mother") >= 0 && statement.indexOf("father") >= 0) {
    response = "Tell me more about your parents.";
}
```

This is the Java equivalent of the statement “(mother is present) and (father is present)”. The only time this entire expression is true is if both words are present. If either word is missing, the expression is false. More generally, if either of the sub-expressions (“mother is present”, “father is present”) is false then the entire expression is false. For multiple conditional expressions joined by logical AND, remember that all the sub-expressions have to be true for the entire expression to be true. Don’t let this confuse you – it works exactly the way we use “and” when we’re speaking to each other whether we have two conditional expressions or fifty.

What do you think happens if the user enters a statement that includes the word “mother” but not the word “father”? Will the expression as a whole be true or false? Do you understand why?

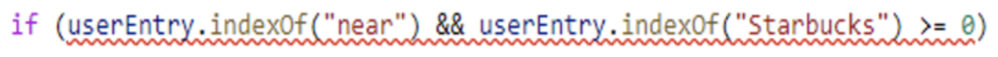
3. Add your code to the getResponse method so it responds with “The nearest Starbucks is in the Student Union 0.5 miles away” when the user’s entry contains the word “near” and the word “Starbucks”. For example, a possible statement and response would be:

    Statement: **Where is the nearest Starbucks**.

    Response: **The nearest Starbucks is in the Student Union 0.5 miles away.**

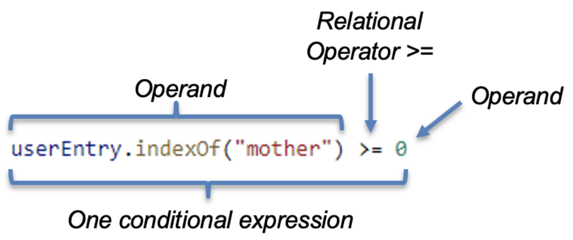
   You must use a single IF statement with a compound conditional.

   Note that you have to be careful the way you construct conditional expressions. You might think this will work:



```java
if (userEntry.indexOf("near") && userEntry.indexOf("Starbucks") >= 0)
```

As you can see from the red squiggly line there is a problem with the way this statement is constructed. Each conditional must be complete by itself: two operands separated by a relational operator:



```
userEntry.indexOf("mother") >= 0
```

4. When you are satisfied with your code, show it to your instructor or a TA to receive credit for this section.

Part B:

In this part we want to practice using the logical operator OR.

1. There are situations where we want our code to respond if any of a number of conditionals are true, not just if all of them are true. For our chatbot, if the user enters a statement like "My sister attends UNCC" or a statement like "I have two brothers", we can have the chatbot respond with "Tell me more about your siblings". We're asking the chatbot to respond the same way if the word "sister" or the word "brother" appears in the user's entry. Again, this works like the word "or" does when we're speaking. With logic rules, if any of the sub-expressions is true then the entire expression is true

   Programming our chatbot with this capability addresses the situations where we would want our chatbot to respond the same when the user mentions something that is a specific type of some category. In our example, "brother" and "sister" can be thought of as a type of sibling. Translating this knowledge we have about types and categories to our chatbot can make it seem smarter!

2. Add to the getResponse code so it responds, "Tell me more about your pets" when the user's entry contains the word "dog" or the word "cat". You must use a single IF statement with a compound conditional. For example, a possible set of statements and responses would be:

   Statement: **I like my cat Mittens**.
   Response: **Tell me more about your pets**.

   Statement: **My dog likes tennis balls**.
   Response: **Tell me more about your pets**.

3. When you are satisfied with your code, show it to your instructor or a TA to receive credit for this section.

Part C:

In this part you will practice using the logical NOT operator (exclamation mark ! in Java) to reverse the truth value of a multiple conditions.

1. You can apply NOT to a single expression. For example, if the following expression is true:

   ```
   if (userEntry.indexOf("graduate") >= 0)
   ```

   you can reverse its truth value by adding the NOT operator:

   ```
   if (!(userEntry.indexOf("graduate") >= 0))
   ```

   Notice that the original condition, `userEntry.indexOf("graduate") >= 0`w enclosed in parentheses. If you apply the NOT operator to only the userEntry part of the conditional you will see an error:

   ```
   if (!userEntry.indexOf("graduate") >= 0)
   ```

   Java applies the NOT operator to the expression that immediately follows it, in this case `userEntry.indexOf("graduate")` which evaluates to an int. This generates the error The operator ! is undefined for the argument type(s) int because an int doesn't have a truth value thus you can't apply the NOT operator to it to reverse its truth value. But if you enclose the original conditional in parentheses the parenthetical expression has a truth value that can be reversed with the NOT operator.

   Remember that any time you want to test the reverse of an expression's truth value just add the NOT operator before the expression and feel free to use parentheses to group expressions together.

2. You can also reverse the truth value of multiple conditionals but be careful. As we saw above, NOT applies to the expression that immediately follows it unless you use parentheses to group expressions then NOT applies to the group in parentheses. Recall from the prepwork that:
   !(false && false) is equivalent to **!(false)** which evaluates to **true**.

   This changes if we don't use parentheses and write our condition like this:

   !false && false which is equivalent to **true && false** and that evaluates to **false**.

   This works just like the negative sign in math: -a + b is different from -(a + b). If in doubt, use parentheses to group your conditional expressions rather than depending on Java's order of operations.

3. In this section, we want to use NOT to help the chatbot differentiate between homonyms. Homonyms are words that have the same spelling or sound, but mean different things. Words like these can be tricky for a chatbot to understand, but through the clever use of conditions we can try to suss out the context the homonym is being used in. Think about these two sentences.

   I had to park my car so far away from my classes this morning.

   We had a small picnic in the park this weekend.

4. The homonym we want to detect here is **park**. In order to help the chatbot differentiate between these two, we're going to use && (AND) and ! (NOT). Consider the following pseudocode.

   - If the user's entry contains the word "park", and does not contain the word "car", respond with "I bet the trees there are lovely this time of year."
   - However, if the user's entry only contains the word "park", respond with "Parking on campus is atrocious and expensive."
5. Add the code that accomplishes the goal described in paragraph 4 above to your getResponse() method. Be sure to test your new conditional statements thoroughly. **You MUST make use of the NOT operator and a compound conditional in this Part**. You will probably want to use an IF-ELSE statement to only test for the word "park" if the test for "park" and "car" fails. Try it with two separate IF statements and see what happens.

6. Add another set of conditionals to test context for the word "rock"
   - If the user's entry includes the word "rock" but not the word "music", respond with "I had a rock collection too!"
   - However, if the user's entry just includes the word "rock", respond with "What is your favorite band?"

   Again, you must use a compound conditional with NOT and an IF-ELSE statement might be useful.

7. When you are satisfied with your code, show it to your instructor or a TA to receive credit for this section.

Part D:

In this Part we will explore how we identify individual words, not just substrings of other words.

1. We want to enhance our chatbot so it only responds to certain keywords if they appear as standalone words, not as a part of other words. Remember in the last lab that the response was the same when the user's entry contained "no" or "know"? This was because we were searching for substrings, not individual words. What if we wanted to be able to tell the difference between "no" and "know" as two separate words? Think of how we tell when a word starts or ends a sentence. The simplest rule we follow is looking for spaces. When we start reading a sentence, we typically start at the beginning of the line and go until we encounter the first space. That indicates to us that the word has ended and a new word will start after the space.

   Using this rule of thumb, we can go through a line of text identifying all the words simply by knowing that a character that comes right after a space indicates the current word has ended and a new word is about to begin. That word ends when we encounter another space and so on. Of course, this doesn't account for punctuation because that's another thing that indicates to us when a word/sentence ends and a new one begins. For this lab we will keep it simple and not worry about punctuation and just handle the condition where the String our chatbot is looking for happens to be the first word in the sentence or somewhere in between. Remember that we're trying to identify individual words so "no" won't be found in "know".

2. The String class has a method that checks whether the string starts with another String. This method is helpful in order for us to tell if the statement the user inputted starts with the word the chatbot is looking for.

`startsWith(String myString)`	Tests if this String starts with the specified String. Returns true if it does and false otherwise.

3. Update the condition that handles the String "no" to be true only if the String no is the first word in the statement entered or it is one of the other words. Again you do not need to worry about it being the last word in the sentence.

4. When you are satisfied with your code, show it to your instructor or a TA to receive credit for this section.

Bonus:

Change the code so it responds when <u>three</u> words like "love" <u>and</u> "mom" <u>and</u> cook" are in the user's entry. For example, a possible user entry and response would be:

Statement: **I love my mom's cooking**.
Response: **I'll bet it's delicious**.

You must use a single IF statement with a compound conditional instead of a series of IF statements or IF-ELSE statements.

Bonus-Bonus:

Change the code for Part D so it also recognizes when a word is at the end of the sentence.

Bonus-Bonus-Bonus:

Change the code for Part D so it also recognizes when a word appears in the sentence but is proceeded with a comma or a period.