CS201 - Assignment 8
Due: Wednesday April 23, at the beginning of class

In this assignment we will play a guessing game (somewhat similar to 20 questions), with the computer doing the guessing—and learning at the same time! In the sample below, the human's responses are shown in red.

```
Welcome to the Animals game!

Shall we play a game? y
Were you thinking of a elephant? n
Doh! What was the animal? cow
What question separates cow from elephant? Does it moo?
What is the correct answer for this animal? y

Shall we play a game? y
Does it moo? y
Were you thinking of a cow? y
Great!

Shall we play a game? y
Does it moo? n
Were you thinking of a elephant? n
Doh! What was the animal? gnat
What question separates gnat from elephant? Is it bigger than a breadbox?
What is the correct answer for this animal? n

Shall we play a game? y
Does it moo? n
Is it bigger than a breadbox? y
Were you thinking of a elephant? n
Doh! What was the animal? whale
What question separates whale from elephant? Does it live in the water?
What is the correct answer for this animal? y

Shall we play a game? n
Bye!
```

Strategy

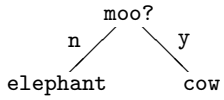
The program maintains a binary tree whose internal nodes contain questions and whose leaves contain the names of animals. The left and right children of an internal node correspond to the responses "no" and "yes" (left being no and right be yes). When the program makes a wrong guess, it collects enough information to create a new node. The original leaf (the one with a wrong answer) is replaced by a new internal node that contains a new question and whose children are the old wrong answer and the new right answer.

For example, we can follow the internal representation for the game above

1. Before the game is played, the tree only has a single node:

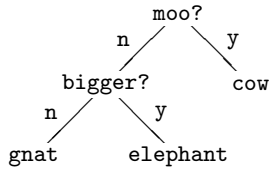
   ```
   elephant?
   ```

2. After the first round of the game, the tree now has a question and two answer nodes:

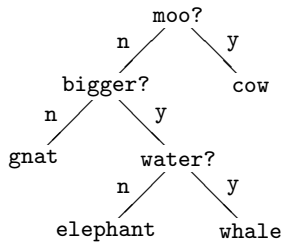


3. After the second round of the game, the tree won't have changed. The tree only changes when the prediction is incorrect.

4. After the third round of the game, we again will add a question node and an additional animal:



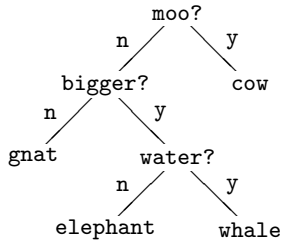
5. Finally, after the fourth round, we will get the final tree of:



A few things to notice:

- The tree has the property that a node is either a leaf or has two children. That property makes it easy to distinguish answers from questions.
- To play a round, we start at the root of the tree and work our way down. At each node, we are asking a question to the user. If it's an internal node, it's a general yes/no questions. If it's a leaf node, it's a guess at the animal the person was thinking of.

So that the games won't start from scratch each time, we're going to store the game tree in a text file. We can do this by listing the nodes of the tree in *preorder* and prepending each line to indicate whether it's an internal node (question node, prefixed with a `Q`) or a leaf node (answer node, prefixed with an `A`). Here is a picture (again) of the final resulting tree and the corresponding textual representation:




```
QDoes it moo?
QIs it bigger than a breadbox?
Agnat
QDoes it live in water?
Aelephant
Awhale
Acow
```

The shortest possible file has one line, an answer.

Animal trees

The central data structure will be a tree of type `AnimalTree` which `extends BinaryTree<String>`. Here are the requirements for the class:

- Do not add any instance variables. The data will all be stored in the parent class's (`BinaryTree`) instance variables. The data instance variable of the parent class can represent either an answer or a question, depending on whether the node is a leaf or not. Similarly, don't encode "Q" or "A" in the actual string.

- You will need one or more constructors, which will likely consist of invocations of the corresponding superclass constructors. Remember, to do this, we use the keyword "`super`", which allows us to call the constructors of the parent class. (See `http://docs.oracle.com/javase/tutorial/java/IandI/super.html` for a refresher on `super`.)
- Include a `public` method `writeFile(String fileName)` that writes the tree to a textfile with the given name. It may be useful to add a recursive auxiliary method.
- You will need another `public static` method `readFile(String fileName)` that creates an `AnimalTree` from a textfile with the given name. The method needs to be static because it returns a new tree. It does not make sense to create a tree and *then* fill it. Again a recursive auxiliary method will likely be useful. You can assume that the textfile is properly formatted, i.e. you don't have to check for erroneous input. This will make your life easier.
- You may not change the `BinaryTree` class at all!

The game

The other class you will write is the `AnimalGame` class. This class will utilize your `AnimalTree` class and will handle the interaction with the user as shown in the example above.

Your class *must* have the following:

- The user's input should come from the console (i.e. `System.in`).
- You should have a `static` method called `play` that takes two `String`s as parameters: the name of the input file to read the starting game tree from and the name output file to write the final game tree to when the user is finished playing the game.
- The class `AnimalGame` is simply a container for static methods. It should have no instance variables or non-static methods.

When `play` is first called it should read in the tree from the file and then start playing the game. As the game is played, the initial tree should be updated, until the user quits. Before the method finishes, it should write out the updated tree.

Hints/Advice

I have included some starter code for you at:

`http://www.cs.middlebury.edu/~dkauchak/classes/cs201/assignments/assign8/`

which has a skeleton of the `AnimalTree` class and the `AnimalGame` class. To make your life easier, I've overridden the `left()` and `right()` methods to type cast them into things of type `AnimalTree`

to make your life easier.[1]

- I'm not asking you to explicitly write any tests, however, I strongly recommend testing incrementally as you go.
- Make sure you understand how everything works in this assignment! As we start to look at more interesting data structures, it's going to become more and more important that you understand exactly what the code you're writing does.
- On a similar note, make sure that you understand what it means for `AnimalTree` to `extend` `BinaryTree`. There are no `protected` instance variables in the `BinaryTree` class, so you will only be able to manipulate the instance variables via the `public` methods.
- Both `writeFile` and `readFile` can be done very reasonably with recursive methods. In both cases, you will likely need an additional helper method. For example, for the `readFile` method, if you created a helper method that took a `BufferedReader` as a parameter, you could read a line from the file and then make recursive calls based on what type of data was on that line (question or answer).
- One easy way to check your read/write methods is to read in a file and then write it back out and make sure you get the same thing.
- To get started, you'll need a pre-existing tree in a file. You can use the tree above, but an even simpler file would be just:

  ```
  Aelephant
  ```

  which would just always guess elephant.

When You're Done

Make sure you have comments at the top of any file/class that you wrote with with your name and the assignment number.

JavaDoc

All of your methods MUST have JavaDoc style headers. I've included it for many of the methods, but make sure you include it for any additional ones you write.

To group these files into a single file, you need to export your project. To do this:

1. Right-click on the project (on Mac, ctrl+click) and select `Export`.

[1] The return type of the `left()` and `right()` from the `BinaryTree` class. Because `AnimalTree` `extends` `BinaryTree`, these methods will be available for the `AnimalTree` class `except` the return type of the methods will still be `BinaryTree`. We know that we're only going to be putting `AnimalTree` objects in our tree, so we can type cast these and override the methods.

2. You'll see a number of options. Open up the `Java` folder and select `JAR file` and click `Next`.

3. You should just see your project selected. Below, make sure **only** the following two options are checked:

   - "Export Java source files and resources"
   - "Compress the contents of the JAR File"

4. Click on the `Browse...` button and pick a location to save the output file. Give the file a name like "kauchak1.jar", where "kauchak" is your last name and "1" is the assignment number.

5. Click `Finish`.

If all went well this will generate a single `.jar` file wherever you picked to save it.

Submit this JAR file as assignment number "8.2" via the online submission mechanism on the course web page.