Compuer Science 62
Assignment 7

Due 11:59pm on Tuesday, March 30, 2010

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
```

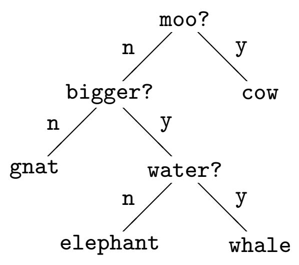
```
Were you thinking of a elephant? n
Doh! What was the animal? whale
What question separates whale from elephant? Does it live in the water?
What is the correct answer for this animal? y

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

This exercise is a warm-up. In a few weeks, we will write the same program in C++[0.13].

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.

Notice that 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. It also allows us to store trees as text files, with the nodes listed in *preorder*. Here is a picture and the corresponding textual representation of a simple tree.



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

The lines in the file follow a preorder traversal of the tree. A question is prefixed by the letter `Q` and an answer by `A`. Do not specify empty trees in the file as they do not provide useful information. 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 single string 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.
- You will need 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.

In addition to the requirements, you may want to override existing `BinaryTree` methods. For example, having `left()` and `right()` return objects of type `AnimalTree` avoids casts in the rest of the program.

Once you have this working, test it to make sure everything is working properly. One way to do this is to read in a file and then write it back out and make sure you get the same thing.

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 `AnimalGame` class will have a main method that should run the animal game (other static methods may also be useful to help you accomplish this).

The user's input will come from the console (i.e. `System.in`). The program will take two command-line paremeters: the first is the filename of the initial animal tree to load and the second is the resulting animal tree after playing the game. As the game is played, the initial tree should be updated, until the user quits. After the games are played and the program is about to terminate, it should write out the updated tree.

The class `AnimalGame` is simply a container for `main` and other static methods. It will have no instance variables or non-static methods.

Make sure you follow this specification exactly. If you need clarification, please feel free to ask me.

Getting started

For consistency use the `BinaryTree` class from bailey's library (i.e. import it from structure5). You can find documentation at:
`http://www.cs.williams.edu/∼bailey/JavaStructures/doc/structure5/index.html`

I've provided a very simple example file and example output from the transcript above at:

`/common/cs/cs062/assignments/assignment7`
but you should test more of your own experiments.

Submission.

Write your code carefully and clearly. Follow the specifications precisely, because your classes may be tested in ways that you do not suspect. As always, we will look at your Javadoc documentation. Submit the final versions of these files, with the names specified, in the usual way.

Grading

You will be graded based on the following criteria:

criterion	points
AnimalTree	5
overall functionality	11
appropriate comments (including JavaDoc)	2
style and formatting	1
submitted correctly	1

Be sure that your input/output match the example above and the data we've provided, particularly make sure that the questions are in the same order, etc. and that you produce and read proper preorder tree files.

Some details

To read files use either the `Scanner` class or the `BufferedReader` class. For output, use either the `BufferedWriter` or `PrintWriter` classes. To communicate with the human at the console, use `System.in`.

This is not a grammar exercise. Your program may ask, for example, "Were you thinking of a elephant?"