CS 158 Lab 17 April 6, 2010

The Animal Guessing Game: an example of decision trees

1. Create a new Java project called Animal Guessing Game. Add a class called **Node** that contains member variables for the left and right subtrees (which are **Node**s) and a **String** that will be used for either a question or a guess. Add constructors and accessors as needed. I would recommend a constructor that takes a **String** argument and puts its value into the guess variable and sets the left and right subtrees to null. Add a method called **isLeaf** that returns **true** if the current node has no children (is a leaf) and **false** otherwise. Add a JUnit test for this method, and make sure it works.

2. Create another class called **Game** that contains a main method. You have to decide how you are going to get input and print output: either use a **Scanner** or a **JOptionPane**. In the main method, start a tree (a single node) that contains only a guess for the initial animal (for example, "dog"). Then print a message welcoming the user to the Animal Guessing Game. Then go into a loop that continues as long as the user wants to keep playing. One iteration through the loop represents the program guessing one animal. The loop should just ask the user to think of an animal, delegate all of the work to a method called **processTree (**that requires the root of the tree as an argument), and then find out whether the user wants to play again. Try to allow for the following different equivalent responses from the user: "yes", "Yes", "y", "Y", "YES", "yeah", "yep".

3. The **processTree** method is recursive. The base case comes up when the node passed in is a leaf, in which case you can call a method **guessAnimal** (to be written later) that takes the leaf node as an argument. If you are not handling a base case, call another method **getAnswer** (to be written later) that also takes the internal node as an argument and returns a response from the user to the yes-or-no question asked in the node. If the answer is yes, recursively process the left subtree. Otherwise, recursively process the right subtree. So remember, the left subtree corresponds to "yes" answers, the right subtree corresponds to "no" answers.

4. The **getAnswer** method is fairly simple. Just print the data that is in the node. It will be a question, since the node is not a leaf, and so the user will supply a yes or no answer. Get that answer from the user and return it.

5. We still have to work on the **guessAnimal** method. Given that it will use a helper method, this method itself is not too hard. Start by asking the user if the animal stored in the argument node is the one the user was thinking of. If it is, just print out an appropriate winning message. Otherwise, call another method, **addQuestion**, which will take the current node as an argument and handle the losing event.

6. For the **addQuestion** method,
   a. announce the loss and find out what animal the user was thinking of
   b. get a new question from the user that will distinguish between the animal the user was thinking of and the animal that is in the node (we ***are*** processing a leaf here; do you see why?)
   c. create two new nodes and hang them on the leaf node that was passed in
   d. find out how the user's animal would answer the new question we just obtained
   e. if the answer is yes, put the user's animal in as data in the left subtree and put the animal in the current node in as data in the right subtree
   f. otherwise put the two animals in the subtree nodes the other way around
   g. put the new question that the user supplied in as data in the current node

   At this point, the current node is no longer a leaf. It has a question in it, and two children that are leaves (guesses of animals).

7. Your program should now work. Run it and make sure it's working logically. Fine tune the dialogue so that it looks nicer.

8. The next step is to be able to save and restore the knowledge your program has acquired. This will take some work, and we'll save it for the next lab. If you want to start on it, the plan is to store in a file a single object that contains the whole tree. To do this, we will use a preorder traversal to put the entire tree into an ArrayList at the end of the program. This will require using FileOutputStream and ObjectOutputStream. Then we'll start the program by restoring the tree from the file, if the user wishes, by reading the ArrayList object back from the file. This will require using FileInputStream and ObjectInputStream. Then we have to remove nodes from the ArrayList and reattach them. This will require some careful thought. After this we should be able to continue with the program as before.