Lab 9

20 Questions

In this lab, you’ll build up from a skeleton to create a program to play “20 Questions”. Download and import the given code from the assignments part of the course webpage http://courses.knox.edu/cs142 (on the “Assignments” page).

A transcript of a short session with this program is as follows:

Think of an object and I will try to guess it.
Is it an animal?
yes
My guess: a bear. Is this correct?
no
What was your object? (include a, an, the, ...)
a buffalo
What is a yes/no question that can distinguish a buffalo from a bear?
Does it eat meat?
What is the correct answer for a buffalo?
no
Would you like to play again?
yes
Think of an object and I will try to guess it.
Is it an animal?
yes
Does it eat meat?
no
My guess: a buffalo. Is this correct?
yes
Yay! I guessed it!
Would you like to play again?
no

Of course, the given code doesn’t do anything this sophisticated. Currently, it just asks one question and then makes a guess between the two things it knows about, a bear and a rock. Clearly, there is a lot to do...

The program is built around a data structure called a decision tree, which is a binary tree where the interior (i.e. non-leaf) nodes have questions. These are the questions that the program asks; depending on how they are answered, the program recurses down either the left or right subtrees (which we call yesSubtree and noSubtree). The leaves of the decision tree represent the guesses that the program makes.

In terms of implementation, we have represented both kinds of nodes with a single Node class. It has one attribute, an Action. Action is an abstract class whose implementations are LeafAction and InteriorAction. Which of these types of Action is stored in a Node determines which kind of Node it is.

Begin by playing the basic game to see how it works. Then modify the Questioner constructor to change the question and answers for the initial tree.

Next, familiarize yourself with the given code. You’ll want to understand the method Questioner.ask since it is useful for asking the user yes/no questions. Also be sure that you understand the parts of an InteriorAction and how its constructor works.
Next write a new constructor for Questioner that takes a Scanner and asks the user for the information needed to create a tree (just one with 2 leaves like the sample one). Specifically, have it ask for two objects (one at a time), a question that it can use to distinguish between them, and then what answer is appropriate for the first object. Follow the style of questions in the transcript. Use the nextLine method of Scanner and Questioner.ask to get the answers to your questions. Then use the answer to create a Node that root can point to.

Test your code by changing the program so that it uses the new constructor in main. Then run the program, building the initial tree with the new constructor and then using it to guess.

Of course, the program becomes much more interesting as the decision tree gets bigger and includes more questions. To do this, we need to modify the play method of LeafAction. This is the code that runs when the program makes its guess. Notice the if statement where the program makes a guess and then prints a message about whether it was correct or incorrect. Replace the message about being wrong with the line

```java
whatItDoes = makeQuestion(input);
```

This will replace the Action of the current Node with an InteriorAction that asks a new question. To do this, write the makeQuestion method, which takes a Scanner and returns an InteriorAction. The method should ask questions and create the InteriorAction in a way very similar to your new Questioner constructor except that it only needs to ask about one object since the object attribute already has one (the guess made by the current LeafAction).

Once you’ve completed those steps, check that your application works. Now you just have to teach it about all the possible objects that users can be thinking about... (If you get bored with that, work toward fixing its other main issue: the tree is destroyed when the program ends. Instead, you could store the decision tree when the program ends by printing a pre-order traversal of the nodes. If you get that working, write a version that reads in such a traversal and builds the tree. (You’ll need to put markers into the output so that you can tell whether the next line of text is the guess for an LeafAction or the question for an InteriorAction.))