CSE143—Computer Programming II
Programming Assignment #6
due: Friday, 8/9/13, 9 pm

This assignment will give you practice with binary trees. Turn in files named `QuestionTree.java` and `QuestionNode.java` from the Homework section of the course web site. You will need the support files `QuestionMain.java` and various sample question files from the Homework section of the course web site; place them in the same folder as your classes. All files can be downloaded together in `hw6.zip`.

Program Description:

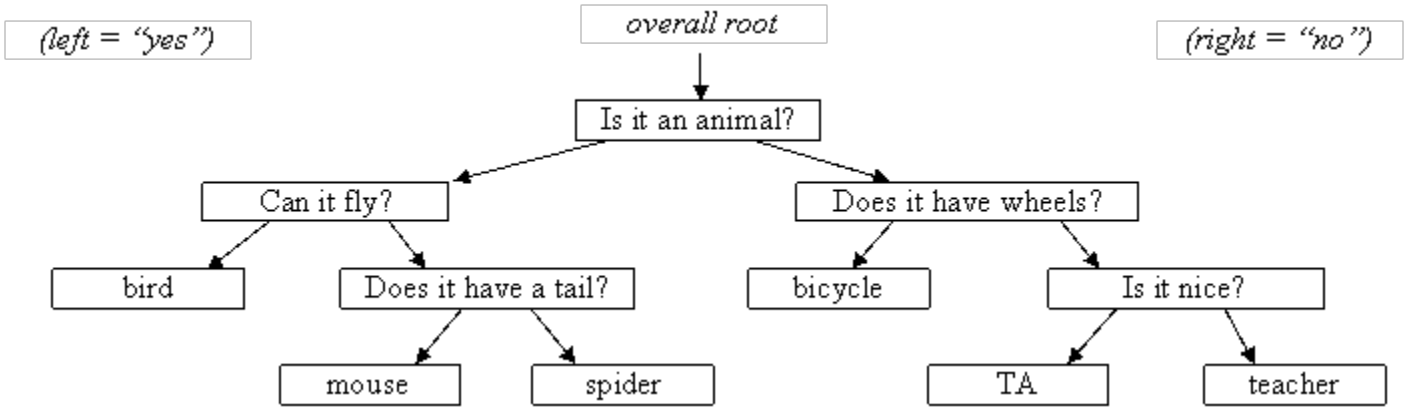
You are to implement a yes/no guessing game. The idea is that you construct a binary tree where each leaf has the name of an object and each branch node has a yes/no question that distinguishes between the objects. It's like a big game of 20 questions where each question is a yes/no question (although the game can have any number of questions).

Each round of the game begins by you (the human player) thinking of an object. The computer will try to guess your object by asking you a series of yes or no questions. Eventually the computer will have asked enough questions that it thinks it knows what object you are thinking of. It will make a guess about what your object is. If this guess is correct, the computer wins; if not, you win.

The computer keeps track of a binary tree whose nodes represent questions and answers. (Every node's data is a string representing the text of the question or answer.) A "question" node contains a left "yes" subtree and a right "no" subtree. An "answer" node is a leaf. The idea is that this tree can be traversed to ask the human player a series of questions.

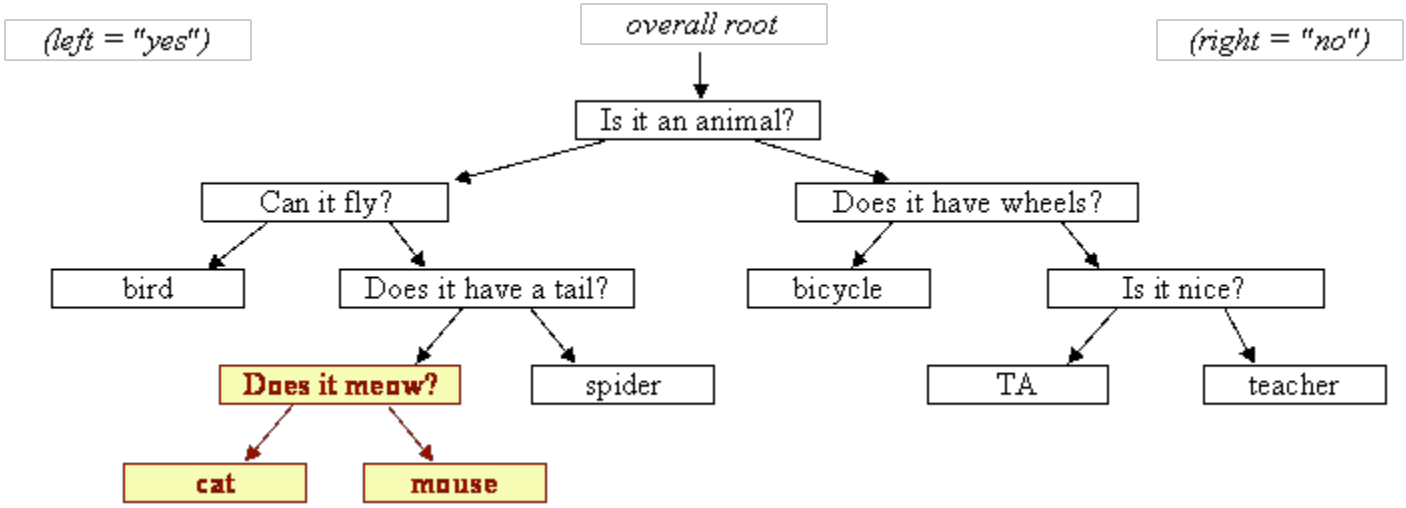
For example, in the tree below, the computer would begin the game by asking the player, "Is it an animal?" If the player says "yes," the computer goes left to the "yes" subtree and then asks the user, "Can it fly?" If the user had instead said "no," the computer would go right to the "no" subtree and then ask the user, "Does it have wheels?"

This pattern continues until the game reaches a leaf "answer" node. Upon reaching an answer node, the computer asks whether that answer is the correct answer. If so, the computer wins.



Initially the computer is not very intelligent, but it grows more intelligent each time it loses a game. If the computer's answer guess is incorrect, you must give it a new question it can ask to help it in future games. For example, suppose that the player is thinking of a cat. The computer will end up at the leaf node for mouse. The computer will guess mouse and the user will say that the guess is wrong. At that point, the computer asks what the user was thinking of ("cat") and a question to distinguish it from a mouse (the user might say, "Does it meow?") and what the answer is.

The computer takes the new information from a lost game and uses it to replace the old incorrect answer node with a new question node that has the old incorrect answer and new correct answer as its children:



Implementation Details:

Your `QuestionTree` class should store the binary tree and should have the following public methods:

`public QuestionTree()`
This method should construct a question tree with one leaf node representing the object "computer".

`public void read(Scanner input)`
This method will be called if the client wants to replace the current tree by reading another tree from a file. Your method will be passed a `Scanner` that is linked to the file and should replace the current tree with a new tree using the information in the file. Assume the file is legal and in *standard format* (see second paragraph below). Read entire lines of input using calls on `nextLine()`.

`public void write(PrintStream out)`
This method will be called if the client wants to store the current tree to an output file. It should write out the tree to the given `PrintStream` in *standard format* (see second paragraph below). The `PrintStream` will be open for writing.

`public void askQuestions()`
In this method you should use the current tree to ask the user a series of yes/no questions until you either guess their object correctly or until you fail, in which case you expand the tree to include their object and a new question to distinguish their object from the others.

`public boolean yesTo(String prompt)`
This method asks the given question until the user types "y" or "n". It returns `true` if "y", `false` if "n". The content of this method is given to you below.

In addition, you must define a class `QuestionNode` for use in your tree. In defining your tree node class, you should decide what data fields you need for solving this particular problem. In other words, you don't have to use a generic binary tree; you can use one that is highly specific to this program. Your node class should have at least one constructor. It can have more than one, but don't include any constructors that you don't actually use in your `QuestionTree` class.

Your program also must be able to write the tree to an output file and read it back in through the `read` and `write` methods. That way your question tree can grow each time a user runs the program. To be able to read and write the tree, we need a set of rules for how to represent the tree, which we'll refer to as the *standard format* for a question tree. A tree is specified by a nonempty sequence of line pairs, one for each node of the tree. The first line of each pair should contain either the text "Q:" to indicate that it is a question node (i.e., a branch node) or the text "A:" to indicate that it is an answer node (i.e., a leaf node). The second line of each pair should contain the text for that node (the question or answer). The nodes should appear in preorder (i.e., in the order produced by a preorder traversal of the tree).

This program involves interaction with the user. As always, we will use a `Scanner` linked to `System.in` to do this. But with interactive programs, you don't want to have more than one `Scanner` linked to `System.in`. For that reason, in this program your class will construct the console `Scanner`. It turns out that all that the main method needs is the ability to ask a yes/no question of the user. So the interface for your class will include the method `yesTo` that main will call when it wants to interact with the user. You must construct a single console `Scanner` that you store in a data field and use throughout your class. All calls to the console `Scanner` should be on the `nextLine` method so that you always read an entire line of user input.

The details of the `yesTo` method are fairly uninteresting, so it is being provided for you (this code assumes a data field called `console` has been initialized). You are to include this method without modification in your `QuestionTree` class.

```
// post: asks the user a question, forcing an answer of "y " or "n";
//       returns true if the answer was yes, returns false otherwise
public boolean yesTo(String prompt) {
    System.out.print(prompt + " (y/n)? ");
    String response = console.nextLine().trim().toLowerCase();
    while (!response.equals("y") && !response.equals("n")) {
        System.out.println("Please answer y or n.");
        System.out.print(prompt + " (y/n)? ");
        response = console.nextLine().trim().toLowerCase();
    }
    return response.equals("y");
}
```

A log of execution is provided at the end of this write-up showing how the program should interact using your `QuestionTree` class. You are to exactly reproduce the format of this log.

You might be tempted in writing this program to "morph" what used to be an answer node of the tree into a question node. This is considered bad style. Question nodes and answer nodes are fundamentally different kinds of data. You can rearrange where they appear in the tree, but you shouldn't turn a former answer node into a question node just to simplify the programming you need to perform.

Style Guidelines and Grading:

Part of your grade will come from the elegance of your recursive algorithm; don't create special cases in your recursive code if they are not necessary or repeat cases already handled. Use the *x=change(x)* idiom appropriately. Redundancy is another major grading focus; you should avoid repeated logic as much as possible. Your class may have other methods besides those specified, but any other methods you add should be `private`.

You should follow good general style guidelines such as: making fields `private` and avoiding unnecessary fields; appropriately using control structures like loops and `if`/`else`; properly using indentation, good variable names and types; and not having any lines of code longer than 100 characters.

Comment your code descriptively in your own words at the top of your class, each method, and on complex sections of your code. Comments should explain each method's behavior, parameters, return, pre/post-conditions, and exceptions.

Sample Execution:

```
Welcome to the cse143 question program.

Do you want to read in the previous tree? (y/n)? n

Please think of an object for me to guess.
Would your object happen to be computer? (y/n)? n
What is the name of your object? dog
Please give me a yes/no question that
distinguishes between your object
and mine--> Is it an animal?
And what is the answer for your object? (y/n)? y

Do you want to go again? (y/n)? y
Please think of an object for me to guess.
Is it an animal? (y/n)? y
Would your object happen to be dog? (y/n)? n
What is the name of your object? frog
Please give me a yes/no question that
distinguishes between your object
and mine--> Does it hop?
And what is the answer for your object? (y/n)? y

Do you want to go again? (y/n)? y
Please think of an object for me to guess.
Is it an animal? (y/n)? y
Does it hop? (y/n)? n
Would your object happen to be dog? (y/n)? y
Great, I got it right!

Do you want to go again? (y/n)? n
```

Sample data file `question.txt` (after above log):

```
Q:
Is it an animal?
Q:
Does it hop?
A:
frog
A:
dog
A:
computer
```