CSE 143: Computer Programming II   Autumn 2016

HW7: 20 Questions (due Thursday, December 1, 2016 11:30pm)

This assignment focuses on binary trees and recursion. Turn in the following files using the link on the course website:

- `QuestionsGame.java` – A class that represents a tree of questions and answers

You will need the support files `QuestionMain.java`, `spec-question.txt`, and `big-question.txt`; place these in the same folder as your program or project. You should not modify the provided files. If you are using EZclipse, the files will be automatically downloaded for you in the correct location. You should not modify the provided files. The code you submit must work properly with the unmodified versions.

The Game of Twenty Questions

20 Questions is a guessing game in which the objective is to ask yes/no questions to determine an object. In our version, the human begins a round by choosing some object, and the computer attempts to guess that object by asking a series of yes/no questions until it thinks it knows the answer. Then, the computer makes a guess; if its guess is correct, the computer wins, and otherwise you win. For example, consider the following game:

```
>> Is it an animal? (y/n)?  n
>> Does it have wheels? (y/n)?  y
>> I guess that your object is bicycle!
>> Am I right? (y/n)?  y
>> Awesome! I win!
```

In this assignment, you will create a class named `QuestionsGame` to represent the computer's tree of yes/no questions and answers for playing games of 20 Questions. Your `QuestionsGame` class **MUST** contain a private static inner class named `QuestionNode`.

You are provided with a client `QuestionMain.java` that handles user interaction and calls your tree's methods to play games.

QuestionNode

The contents of the `QuestionNode` class are up to you. Though we have studied trees of `int`s, you *should* create nodes specific to solving this problem. Your `QuestionNode` class *must* be `private static` inner class within `QuestionGame`. Your `QuestionNode` class should have at least one constructor used by your tree. Your node's fields *must* be public. Any fields representing data *must be final*. `QuestionNode` should not contain *any actual game logic*. It should only represent a single node of the tree.

QuestionsGame

This class represents a game of 20 Questions.

It 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.)

The *leaves* of the tree represent possible answers (guesses) that the computer might make. All the other nodes represent questions that the computer will ask to narrow the possibilities. The left branch indicates the next question the computer asks if the answer is *yes*, and the right branch is the next question if the answer is *no*.

Note that even though the name of the game is "20 questions", the computer will *not* be limited to only *twenty*; the tree may have a larger height.

`QuestionsGame` **should have the following constructors:**

public **QuestionsGame**(String object)
This constructor should initialize a new `QuestionsGame` object with a *single leaf node* representing the object `object`. You may assume the `String` is not `null`. To get `QuestionMain` to use this constructor, you should supply an empty file or one that doesn't exist.

public **QuestionsGame**(Scanner input)
This constructor should initialize a new `QuestionsGame` object by reading from the provided `Scanner` containing a tree of questions in standard format. You may assume the `Scanner` is not `null` and is attached to a legal, existing file in standard format. Make sure to read entire lines of input using calls on `Scanner`'s `nextLine`.

`QuestionsGame` **should also implement the following methods:**

public void **saveQuestions**(PrintStream output)
This method should store the current questions tree to an output file represented by the given `PrintStream`. This method can be used to later play another game with the computer using questions from this one. You should throw an `IllegalArgumentException` if the `PrintStream` is `null`.

public void **play**()

This method should use the current question tree to play one complete guessing game with the user, asking yes/no questions until reaching an answer object to guess. A game begins with the root node of the tree and ends upon reaching an answer leaf node.
 	
If the computer wins the game, this method should print a message saying so.
 	
Otherwise, this method should ask the user for the following:
 	
   • what object they were thinking of,
   • a question to distinguish that object from the player's guess, and
   • whether the player's object is the yes or no answer for that question.

User Input: Yes and No

At various points in this assignment, you will need to get a yes or no answer from the user. You should consider any word that begins with "y" to be "yes" (Yes, YEs, Yup, yeehaw), and anything else to be a no. The easiest way to do this (assuming `console` is a `Scanner` initialized to `System.in`) is the following:

`console.nextLine().trim().toLowerCase().startsWith("y")`

Question Tree "Standard Format"

In `saveQuestions` and the second constructor, your class will be interacting with files containing questions. Just like with BNF where we used a common format so everyone could share the same grammars, here, we specify a common question tree format.
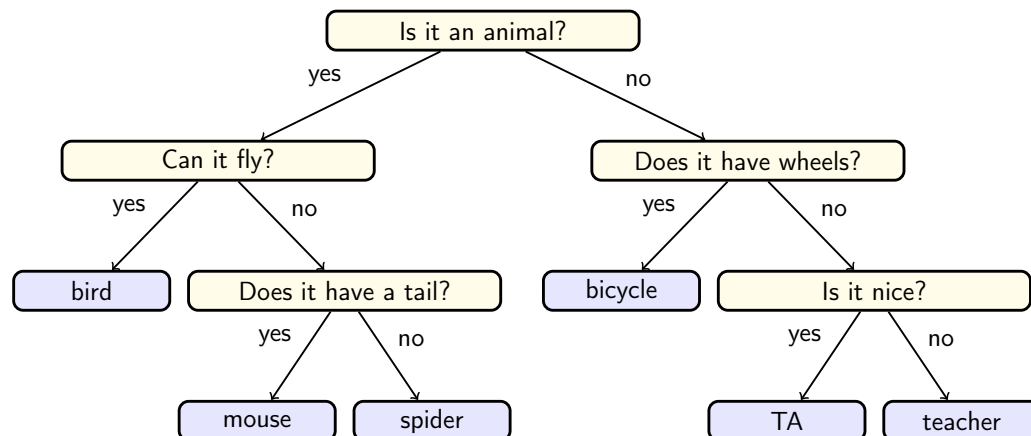
A single `QuestionNode` should be represented as a non-empty sequence of line pairs. The first line of the pair should contain either "Q:" or "A:" to differentiate between questions (branches) and answers (leaves). The second line of the pair should contain the text for that node (the *actual* question or answer).

The nodes of the tree should appear in *pre-order*. The `readTree` and `writeTree` methods from section will be *very helpful* for any method that deals with this format.

Full Example Walk-Through

```
┌─ spec-questions.txt ─┐
│ Q:                   │
│ Is it an animal?     │
│ Q:                   │
│ Can it fly?          │
│ A:                   │
│ bird                 │
│ Q:                   │
│ Does it have a tail? │
│ A:                   │
│ mouse                │
│ A:                   │
│ spider               │
│ Q:                   │
│ Does it have wheels? │
│ A:                   │
│ bicycle              │
│ Q:                   │
│ Is it nice?          │
│ A:                   │
│ TA                   │
│ A:                   │
│ teacher              │
└──────────────────────┘
```


The following output log shows one game being played on the above tree:

```
>> Welcome to CSE 143 Game of N-Questions!
>>
>> Which questions file would you like to use? spec-questions.txt
>> Let's play!  Please choose your object, and I'll start guessing.
>> Press Enter when you're ready to begin!
>>
>> Is it an animal? (y/n)?  n
>> Does it have wheels? (y/n)?  y
>> I guess that your object is bicycle!
>> Am I right? (y/n)?  y
>> Awesome! I win!
>>
>> Do you want to play again (y/n)?  n
```

Initially, the computer is not very intelligent, but it grows smarter each time it loses a game. If the computer guesses incorrectly, it asks you to give it a new question to help in future games. For example, suppose in the preceding log that the player was thinking of a car instead. You might get this game log:

```
>> Welcome to CSE 143 Game of N-Questions!
>>
>> Which questions file would you like to use? spec-questions.txt
>> Let's play!  Please choose your object, and I'll start guessing.
>> Press Enter when you're ready to begin!
>>
>> Is it an animal? (y/n)?  n
>> Does it have wheels? (y/n)?  y
>> I guess that your object is bicycle!
>> Am I right? (y/n)?  n
>> Boo! I Lose.  Please help me get better!
>> What is your object? car
>> Please give me a yes/no question that distinguishes between car and bicycle.
>> Q: Does it get stuck in traffic?
>> Is the answer "yes" for car? (y/n)?  y
>>
>> Do you want to play again (y/n)?  n
```

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.

After the preceding log, the computer's overall game tree would be the following:

```
─────── spec-questions.txt ───────
Q:
Is it an animal?
Q:
Can it fly?
A:
bird
Q:
Does it have a tail?
A:
mouse
A:
spider
Q:
Does it have wheels?
Q:
Does it get stuck in traffic?
A:
car
A:
bicycle
Q:
Is it nice?
A:
TA
A:
teacher
```
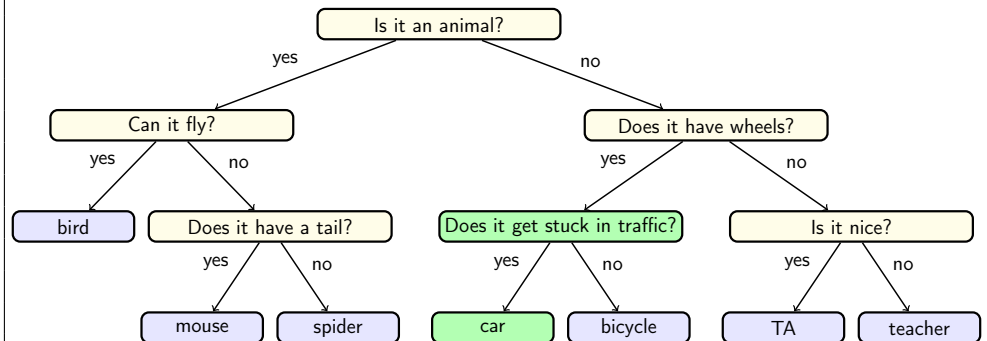

Note that, as usual, your output must match the output in the specification exactly. The output comparison tool will be helpful to make sure that your output is identical.

Creative Aspect (`myquestions.txt`)

Along with your program, turn in a file `myquestions.txt` that represents a saved question tree in the format specified from your `saveQuestions` method. For full credit, this must be in proper format, have a height of at least 5, and be your own work.

Development Strategy

We suggest that you develop the program in the following stages:

(1) Before you begin, you should "stub" the methods you intend to write. In other words, you want to be able to test using `QuestionsMain`; so, you should provide "dummy" methods that do nothing for it to call.

(2) First, you should decide what fields belong in the `QuestionNode` and `QuestionsGame` classes. Once you've chosen the fields, you should implement the *first* constructor (the one that takes a *String*).

(3) Next, you should implement `saveQuestions` (because it's easier than loading them). Make sure to look back at `writeTree` from the section handout!

(4) Then, you should implement the second constructor which reads in a question tree. Make sure to look back at `readTree` form the section handout!

(5) Finally, you should implement `play`. At this point, you'll be able to play the game, even though you aren't able to save or load any questions. When you play, you can add questions one by one and play with your game to check if it's working.

Style Guidelines and Grading

Part of your grade will come from appropriately utilizing binary trees and recursion to implement 20 questions as described previously. Every method with complex code flow should be implemented recursively, rather than with loops. A full-credit solution must have zero loops. We will also grade on the elegance of your recursive algorithm; don't create special cases in your recursive code if they are unnecessary.

x = change(x)

An important concept introduced in lecture was called `x = change(x)`. This idea is related to proper design of recursive methods that manipulate the structure of a binary tree. You must follow this pattern on this assignment to receive full credit on this assignment.

For example, at the end of a game lost by the computer, you might be tempted to "morph" what used to be an answer node of the tree into a question node. Since the data in your question node must be final, this should not compile. One of the reasons this is the case is question nodes and answer nodes are fundamentally different kinds of data. You can rearrange where nodes 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.

Avoid Redundancy

Create "helper" method(s) to capture repeated code. As long as all extra methods you create are `private` (so outside code cannot call them), you can have additional methods in your class beyond those specified here. If you find that multiple methods in your class do similar things, you should create helper method(s) to capture the common code.

Generic Structures

You should always use generic structures. If you make a mistake in specifying type parameters, the Java compiler may warn you that you have "unchecked or unsafe operations" in your program. If you use jGRASP, you may want to change your settings to see which line the warning refers to. Go to `Settings/Compiler Settings/Workspace/Flags/Args` and then uncheck the box next to "Compile" and type in: `-Xlint:unchecked`

Data Fields

Properly encapsulate your objects by making your fields `private`. (You can and *should* make your fields in `QuestionNode` public.) Avoid unnecessary fields; use fields to store important data of your objects but not to store temporary values only used in one place. Fields should always be initialized inside a constructor or method, never at declaration.

Java Style Guidelines

Appropriately use control structures like if/else statements. On this specific assignment, you should not need to use any loops. Avoid redundancy using techniques such as methods and factoring common code out of if/else statements. Properly use indentation, good variable names, and types. Do not have any lines of code longer than 80 characters.

Commenting

You should comment your code with a heading at the top of your class with your name, section, and a description of the overall program. All method headers should be commented as well as all complex sections of code. Make sure you describe complex methods inside methods. Comments should explain each method's behavior, parameters, return values, and assumptions made by your code, as appropriate. The `ArrayIntList` class from lecture provides a good example of the kind of documentation we expect you to include. You do not have to use the pre/post format, but you must include the equivalent information – including the type of exception thrown if a precondition is violated. Write descriptive comments that explain error cases, and details of the behavior that would be important to the client. Your comments should be written in your own words and not taken verbatim from this document.