Game of Twenty Questions

Collaboration: Complete this by yourself. You may get help from Section Leaders or Rick

Credits: This project is one of several resulting from the Apprenticeship learning approach of Owen Astrachan (Duke), Robert Smith (North Carolina Central), James Wilkes (Appalachian State), and their students with support in part by the National Science Foundation Grant #DUE-9954910. This assignment provides practice with trees, recursion, file input, file output, and a linked hierarchical data structure--specifically, a binary tree.

Game of Twenty Questions: The game of *Twenty Questions* can be amusing, especially if the domain of questions is one that you find interesting. For this assignment you will write a program that permits the user to play a dynamic version of twenty questions -- the user can add new questions and answers. You are asked to invent your own questions and answers. The detailed dialog below deals with animals, but you may develop your own theme and your own questions and answers.

Details: In order to run this game with our GUI, we have designed a `GameTree` type. The beginning of the GameTree class is provided to ensure all methods have correct method signatures. First get a start to all files needed for this project from http://www.cs.arizona.edu/people/mercer/Projects/GameOf20Start.zip

Choice.java `enum Choice` lets us use `Choice.Yes` and `Choice.No` rather than case sensitive strings

GameTree.java `public class GameTree` has all methods written as stubs. Each method is commented.

```
- public GameTree(String name)
- public String toString()
- public void add(String newQuestion, String newAnswer)
- public boolean foundAnswer()
- public String getCurrent()
- public void playerSelected(Choice yesOrNo)
- public void reStart()
- public void saveGame()  // Do this method last
```

GameTreeSmallTest.java The beginning of a unit test with a commented `testSaveGame` method

Recommended: Get these three methods working first:

1. Constructor that takes a file name and should call a recursive method to build the binary tree in a pre order fashion.
2. `getCurrent()` to return the data at the current node
3. `playerSelected(Choice yesOrNo)` which is way to go left or right down the tree

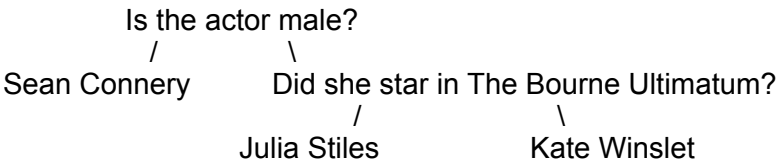
File input: The questions and answers must be stored in a binary tree data structure with `String` elements (no generics necessary). We also ask that you add a constructor that uses the build method that reads from an input file to build the game tree. Input consists of a text file with questions and answers. The input file uses the following schema:

Question?
Yes Answer (left subtree)
No Answer (right subtree)

The input file must have "?" at the end of questions and the answers must not have "?" at the end of the answer. For example actors.txt may have these two questions and three answers:

Is the actor male?
Sean Connery
Did she star in The Bourne Ultimatum?
Julia Stiles
Kate Winslet

This would create a binary tree with the root node at the top representing the first question (going left means yes left, going right means no).



The `toString` method should return a string that looks like this (prepend the string `"- "` for each level)

```
- - Kate Winslet
- Did she star in The Bourne Ultimatum?
- - Julia Stiles
Is the actor male?
- Sean Connery
```

Reading an input file containing text: Use the `Scanner` class to read from the input file. You will need these three imports so the file can be constructed in the `GameTree` constructor inside the `try` block.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class GameTree {

  // Need an inner TreeNode class here . . .
  private TreeNode root;
  private Scanner inFile;
  private String currentFileName;

  public GameTree(String fileName) {
    currentFileName = fileName;
    try {
      inFile = new Scanner(new File(currentFileName));
    }
    catch (FileNotFoundException e) {
      // This block would execute if currentFileNane is not found.
      // We will not have tests to construct GameTree objects with non-existent files.
    }
    root = build();
    infile.close();
  }
}
```

In the `build()` method. use Scanner's **`nextLine()`** to return all characters in a line up to, but not including, the end of line. Important: trim the line with a `trim()` message just in case you have an extra blank at the end.

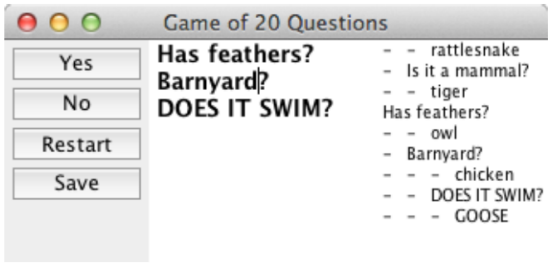
Complete the `saveGame` method last! Once you can play the game with file input where new questions and answers can be added, save the questions and answers to a file. This means the next time you begin the program, the new questions and answers will be there. This allows you to grow the game. Our GUI will have a prompt to see if you want to save the questions and answers. For example, with animals.txt as the input file,

Has feathers?
Barnyard?
chicken
owl
Is it a mammal?
tiger
rattlesnake

and `playerSelected(Choice)` messages are sent until `getCurrent()` returns the "chicken" answer, then when the following message is sent,

```
theGame.add("DOES IT SWIM?", "GOOSE");
```

then the saved file must include this new question and answer to the left (preorder traversal of the tree). The `toString()` version of the `GameTree` should look like the text to the right in the GUI.



Recommended: Use this beginning of a unit test GameTreeSmallTest.java that has the following two methods plus a test method commented out that will test save at the end.

```
  // This code executes once and only once before and @Test method runs.  Good for
initialization.
  // This allows you to ALWAYS begin with the same exact questions and answers
  @BeforeClass
  public static void setUp() throws FileNotFoundException {
    // Let outfile be an object like System.out. You can send outfile println messages
    PrintWriter outFile = new PrintWriter(new FileOutputStream("t2UofATEST.txt"));

    // Write seven lines to an output file:
    outFile.println("Has feathers?");
    outFile.println("Barnyard?");
    outFile.println("chicken");
    outFile.println("owl");
    outFile.println("Is it a mammal?");
    outFile.println("tiger");
    outFile.println("rattlesnake");

    outFile.close();  // close() the file or risk having no text in t2UofATEST.txt
  }

  @Test
  public void testGameWithSevenNodes() {
    // This test uses the input file t2UofATEST.txt with the text shown in setup()
    GameTree aGame = new GameTree("t2UofATEST.txt");
    // This string should be returned from toString()
    // - - rattlesnake
    // - Is it a mammal?
    // - - tiger
    // Has feathers?
    // - - owl
    // - Barnyard?
    // - - chicken
    assertFalse(aGame.foundAnswer());
    assertEquals("Has feathers?", aGame.getCurrent());
    assertFalse(aGame.foundAnswer());
    Choice userSelection = Choice.Yes;
    aGame.playerSelected(userSelection);
    assertEquals("Barnyard?", aGame.getCurrent());
    assertFalse(aGame.foundAnswer());
    aGame.playerSelected(Choice.No);
    assertEquals("owl", aGame.getCurrent());
    assertTrue(aGame.foundAnswer());
  }
```

Writing Text to a File: To write text to a disk file, use the `FileWriter` and `PrintWriter` classes. Here is some sample code that writes data to the file named output.data

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class TextFileOutput {
  public static void main(String[] args) {
    String outputFileName = "output.data";
    FileWriter charToBytesWriter = null;
    try {
      charToBytesWriter = new FileWriter(outputFileName);
    }
    catch (IOException ioe) {
      System.out.println("Could not create the new file: " + ioe);
    }
    PrintWriter diskFile = new PrintWriter(charToBytesWriter);
    // diskFile understands the familiar print and println methods of System.out
    diskFile.print("First line has one number: ");
    diskFile.println(123);
    diskFile.println(4.56);
    diskFile.println('c');
    diskFile.println(3 < 4);

    // Explicitly close file! If you don't you end up with an empty file.
    diskFile.close();
  }
}
```

The above program creates the following file name output.data in the project

```
First line has one number: 123
4.56
c
true
```

Optional: Run completed game in http://www.cs.arizona.edu/people/mercer/Projects/GameOf20GUI.java

Grading Criteria

+10 Style/Readability

+2 You included your name as a comment in all files
+2 You have a 1 or 2 sentence description of the purpose of each class
+2 The source code is formatted nicely (in Eclipse use Source > Format)
+2 Used meaningful identifiers
+2 All methods are commented with an accurate description in LinkedPriorityLis<E>;

Problem and Code Coverage 90pts

To get 100% for these 90 points, you will need 100% problem coverage and 100% code coverage. -1 for every WebCat submission more than 10, up to -30 points. Also, *you can get a score of 0 because*

- *WebCat reports a compile time error (look for Unknown symbol).*
- *One of your assertions failed on WebCat (even though it passed for you locally)*
- *A WebCast test caused your code to throw any exception*