Twenty Questions

20 Questions is a guessing game in which the objective is to ask yes/no questions to determine a "secret" object. In our version, the computer attempts to guess that object by asking a series of yes/no questions until it thinks it knows the answer.

The computer keeps track of a binary tree whose nodes represent questions and answers. Every node's `data` element is a String representing the text of the question or answer. A "question" node contains a left "yes" sub-tree and a right "no" sub-tree. An "answer" node is a leaf. The idea is that this tree can be traversed to ask the human player a series of questions. (Though the game is called "20 Questions" our game will not limit the tree to a height of 20. Any height is allowed.)

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 a bicycle!
>> Am I right (y/n)?  y
>> Awesome! I win!
```

The game 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 20 questions - the user can add new questions and answers. Every time the computer loses, it gets smarter. If the computer's answer guess is incorrect, you must give it a new question it can ask to help it in future games.

To run this game with the GUI, you are provided a stub version of the GameTree type, an abstraction of the game along with the data structure to store the questions / answers (as a binary tree). Note that GameTree is a "regular" binary tree, not a binary *search* tree.

	`GameTree(String fileName)`	Construct game tree from previously saved text file (that stores the game in a pre-order fashion)
`void`	`add(String newQ, String newA)`	Add a new question / answer pair to the tree
`String`	`getCurrent()`	Get the current question node (where the user is in the game)
`String`	`toString()`	Make a textual representation of the state of the game
`boolean`	`foundAnswer()`	Returns whether or not the computer has reached an answer node (not another question)
`void`	`playerSelected(Choice yesOrNo)`	Update the state of the game tree, given the user's response to a question
`void`	`saveGame()`	Save the state of the game to disk

In addition to GameTree, you are provided with the following files:

`Choice.java`	The `enum` Choice lets us use `Choice.Yes` and `Choice.No` rather than case sensitive Strings or `char`s
`GameTreeTest.java`	A short JUnit test (with a commented `testSaveGame` method) that shows you how to test your game (you could also just print the results to the console and compare them visually, however JUnit is a very nice testing framework you should learn at some point). **More info on JUnit to come.**
`GameOf20GUI.java`	A simple GUI for playing your game
`"animals.txt"`, etc.	Some text files for testing (feel free to make your own too)

It is recommended to get these three methods working first (read on for more info):

- GameTree's constructor that loads the contents of the file represented by `fileName` and recursively builds the binary tree in a pre-order fashion
- `getCurrent` (to return the data at the "current" node, for the GUI to display the question/answer)
- `playerSelected(Choice yesOrNo)` (which way to go, left or right down the tree)

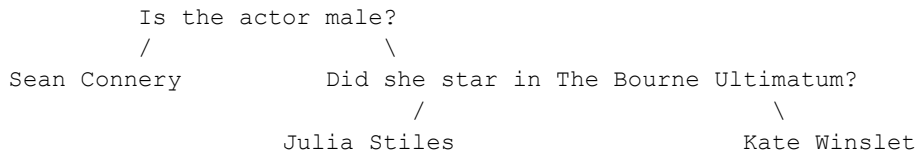
File input. The questions and answers must be stored in a binary tree data structure with String elements (no generics necessary). The input file uses the following schema:

```
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" has 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, going right means no).



The `toString` method should return a String that looks like this (prepend literal `"- "` for each level):

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

Use the Scanner class' `nextLine` method to read from the input file. Use a `try… catch` block (rather than just throwing the exception) to initialize the Scanner. **Important:** trim the line with `trim` just in case you have an extra blank at the beginning or end. Question nodes will `endsWith` a literal "?".

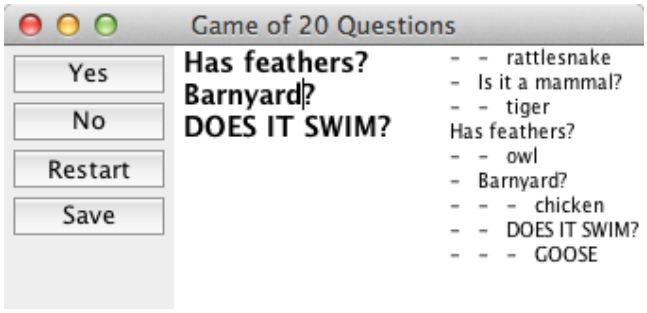
Complete the `saveGame` method last! Once you can play the game with file input where new questions and answers can be added, save the questions and answers to a file. This means the next time you begin the program, the new questions and answers will be there. This allows you to grow the game. Our GUI 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
```

... suppose `playerSelected(Choice)` messages are sent until `getCurrent` returns the "chicken" answer, which the user responds "no" to. The following message will be sent:

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

... the file must include this new question and answer to the left (using a pre-order traversal of the tree). The `toString` version of the GameTree should look something like the text to the right in the GUI.



Use the unit test file **GameTreeTest.java** to test that your code is working; uncomment the `testSaveGame` when you're ready to test the `save` method. The JUnit tests should work by default in BlueJ; the JUnit library is included in Eclipse too, but you'll need to add it to your project's *build path*. Follow these instructions:

- Right-click your project folder
- Select *Build path -> Add libraries…*
- Choose "JUnit", click *Next* then *Finish*

It may help to see the game / GUI in action; see video linked in Canvas.

Writing Text to a File. To write text to a file on disk, use the PrintWriter class. Here is some sample code that writes data to the file named "output.data":

```
String outputFileName = "output.data";
PrintWriter diskFile = null;
try {
```

```
        diskFile = new PrintWriter(new File(outputFileName));
    }
    catch (IOException io) {
        System.out.println("Could not create file: " + outputFileName);
    }
    //diskFile has the familiar print and println methods of System.out
    //see the PrintWriter class' API for more info
    diskFile.print("First line has one number: ");
    diskFile.println(123);
    diskFile.println('c');
    diskFile.println(3 < 4);
    //MUST explicitly close file! You will lose all data and end up with an empty file if not
    diskFile.close();
```

The above program creates the file "output.data" in the project directory with the contents shown below. Make sure you close the PrintWriter! If not, the buffer won't flush, and your text file won't have anything in it. Without specifying an absolute or relative path, it will default to creating the file in the project root (the main project folder in Eclipse). You'll have to refresh Eclipse after the file is created for it to show up in the GUI.

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

Once your code works with the smaller data sets, test it with the GUI and the file "animals_long.txt", which comes from the (very smart) Animal Game web site at animalgame.com. You must comment out the 3 `toString()` calls in the GUI to build really large trees, it can't handle it otherwise.