Objective

To implement a binary tree (but not a binary SEARCH tree), and to use it to store knowledge for a program that plays “20 questions”.

20 Questions

20 Questions is a game played by two players.
One player (the answerer) thinks of a thing (an object).
The other player (the questioner) is allowed to ask up to 20 yes/no questions to figure out what object the answerer was thinking of.
The answerer must honestly answer the questions.
Here’s the beginnings of a game played by player 1 (questioner) and player 2 (answerer):

Question 1: Is it an animal?
Answer: No.

Question 2: Is it a plant?
Answer: No.

Question 3: Is it an inanimate object?
Answer: Yes.

Question 4: Is it bigger than a loaf of bread?
Answer: Yes.

Question 5: Is it in this house?
Answer: No.

Question 6: Can I walk to this thing in 10 minutes or less?
Answer: No.

Question 7: Have I seen it?
Answer: Yes.

Question 8: Is it made by humans?
Answer: Yes.

Question 9: Is it in US ?
Answer: Yes.

Question 10: Is it bigger than a human?
Answer: Yes.

Question 11: Is it a building?
Answer: Yes.

Question 12: It is a place people go for entertainment?
Answer: No.

Question 13: Is it west of the US River?
Answer: Yes.

Question 14: Is it south of the VS River?
Answer: Yes.

At this point, they got interrupted by a family emergency, so they didn't end up finishing the game. (The answerer had chosen: The US Pavilion at the Lee University)

In this assignment, your program will be the questioner, and the person who runs the program (the "user") will be the answerer.

Assignment Question

The assignment is divided into three levels. Each level provides a working game, but with more functionality that the previous level. To get full marks for this part of assignment, you need to complete all three levels.

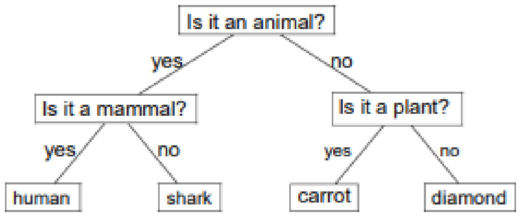
Level 1: A Really Dumb Questioner

Your Questioner will use a binary tree to store knowledge to help it figure out what thing the answerer has thought of - the tree is a **decision tree**.

In a decision tree, a node either has exactly two children - it is an internal node - or it has zero children - it is a leaf. Either way, for your Questioner, a node will contain a String.

- Each internal node will contain, as its String item, a yes/no question.
- Each internal node will have two children:
  - The left child will correspond to the "Yes" answer to the question, and
  - The right child will represent the "No" answer.
- Each leaf contains, as its String item, an answer.

An example of a binary tree representing a decision tree is given below.



The decision tree is used by the Questioner to represent the current knowledge that the Questioner has while it is playing 20 Questions.

For one round of 20 Questions, your Questioner will begin at the root, and move down the tree until it reaches a leaf. At each node, your Questioner does the following:

- If the current node is an internal node, your Questioner will display to the user the question stored in the internal node.
- When the user answers the question, your Questioner will move to the child of the current node that matches the user's answer.

If the current node is a leaf, then your Questioner will present the guess stored in the leaf to the user:

"Are you thinking of a / an _________?"

At this point the Questioner ends the round.

[illegible], le. Create a public class with a name matching your file name - this class will contain main() and any helper methods it calls. This class will create a Questioner and initiate a round of 20 Questions (more details below), but another class will implement a round of 20 Questions (see next paragraph).

Add a second class to your file, the Questioner class. The Questioner class must contain:

- A *private DTNode* (decision tree node) class containing three public instance variables item (a String) and left and right (both DTNodes). The DTNode class should also contain two constructors:
    - A constructor that is passed only a String. This constructor must initialize the new DTNode to contain the String passed to it and make this DTNode into a leaf.
    - A constructor that is passed a String and pointers to two DTNodes. This constructor must initialize the new DTNode to contain the String passed to it and make this DTNode into an internal node whose children are the two DTNodes passed to the constructor.
    - An instance method isLeaf that returns true if the calling DTNode is a leaf (has no children) or false if it is an internal node.

  The DTNode class should not contain anything else.

- A pointer to the root of a decision tree of DTNodes. This decision tree stores the knowledge that the Questioner uses to figure out what thing the Answerer (the user) is thinking of.
- A constructor with no parameters that constructs a small (hardcoded) decision tree for the Questioner to use. (You can create the decision tree pictured above, or create your own. Keep it small, however.)
- A method playRound with no parameters.

  This method works its way down the decision tree in a loop, starting from the root and ending at a leaf.

  If it is at an internal node, the method should ask the question stored in the internal node and get the user's response (always "yes" or "no"). If the user's response is "yes", then the method should move to the left child of the current node. If the user's response is "no", then the method should move to the right child of the current node.

  If it is at a leaf, the method should present the guess stored in the leaf to the user using the following

  phrase: "Are you thinking of a / an ---------------?" and get the user's response (always "yes" or "no"). Then the method should respond with an appropriate message, depending on whether it guessed correctly or not | something like "I guessed correctly!" or "I guessed wrong."

After reaching a leaf, the method should end the loop and return.

To display a yes/no question in a dialog box and to get the user's yes / no response (the user clicks on either a "yes" button or a "no" button), use JOptionPane.showConfirmDialog as follows:

```
int currAnswer
     = JOptionPane.showConfirmDialog(null,
                                     "<question text goes here>",
                                     "20 Questions",
                                     JOptionPane.YES_NO_OPTION,
                        JOptionPane.QUESTION_MESSAGE);
```

After this statement is executed, variable currAnswer will contain either the class constant JOptionPane.YES_OPTION or the class constant JOptionPane.NO_OPTION. Thus, you just need to compare currAnswer to the ap-propriate class constant to gure out what the user responded with (yes or no). You will need to import javax.swing.JOptionPane at the top of your le to be able to do these things.

You can display a message without requiring a user response as follows:

```
JOptionPane.showMessageDialog(null, "<message text goes here>");
```

Finally, the main() method in class should do the following:

- Display an appropriate opening message, such as "Let's play 20 Questions!!!";
- Create a Questioner instance and call playRound() on it.
- Display an appropriate closing message, such as "20 Questions ends".

If you want to, you can restrict the domain that the user can pick from by having method playRound() display a message such as "Think of a movie" before starting the round.

Level 2: A Learning Questioner

The Level 1 Questioner has only a small decision tree of knowledge and doesn't get any better at guessing. In Level 2, the user will be allowed to play multiple rounds of 20 Questions - the program will only stop when the user wants to quit, not after only one round.

Furthermore, when the Questioner does not guess the user's thing, the Questioner will prompt the user to help improve the decision tree.

Start with the Level 1 program and make the following changes:

- Change the main() method (or its helper method) so that it calls playRound() on the Questioner instance in a loop. After each round, the user should be asked if they want to play another round (prompt the user for a yes / no answer with some message such as "I get better every time I play. Play again?"). The loop should continue until the user doesn't want to play another round of 20 Questions.

- Change the playRound() method in the Questioner class as follows.

  If the Questioner guesses incorrectly when it reaches a leaf, then the playRound() method should do the following:

  - Ask the user what they were actually thinking of and get their reply.

    To display a prompt and read a user's reply as a String, do the following:

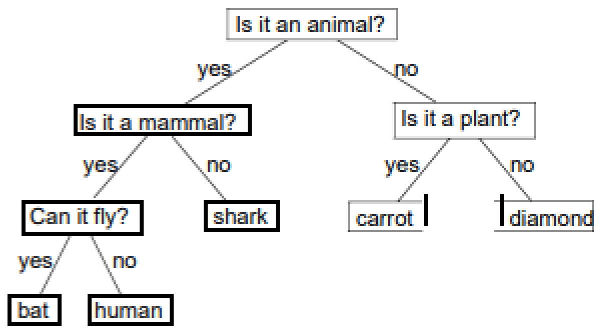
    ```
    String newThing = JOptionPane.showInputDialog("What were you thinking of?");
    ```

  - Then ask for a Yes/No question that would distinguish between the Questioner's wrong guess (e.g., "human") and what the user was thinking of (e.g., "bat") - use a similar line of code here as you did to find out what the user was thinking of. In this example, the user might give as a distinguishing question "Can it fly?"

    Note that the question must always be phrased consistently so that the user's new item is the yes answer to the new question and the Questioner's wrong guess is the no answer to the new question.

    The decision tree must now be modified. The leaf containing the Questioner's wrong guess should be unlinked from its parent, and a new internal node should be inserted in its place. The internal node should contain the new question, and its children should be two leaves for the two answers (the user's new answer and the Questioner's wrong guess).

    For example, the sample hardcoded decision tree given above in Level 1 would be changed to the following:



Level 3: Saving Knowledge

The 20 questions playing software above is fine, but pretty dumb. It must start learning each time you start the software. To solve this problem, write two methods in the Questioner class:

public void writeTree()

This method should prompt the user for a file name and save the current decision tree to the file using the format described below.

public void readTree()

This method should prompt the user for a file name and read a tree from the file.

Each method should have a recursive helper method, since (as described below), they are writing or reading a decision tree in a pre-order traversal.

The format for storing a decision tree is as follows. For the original decision tree shown in Level 1, the result of writing it to the file is the following:

```
< Is it an animal?
< Is it a mammal?
< human
>
< shark
>
>
< Is it a plant?
< carrot
>
< diamond
>
>
>
```

That is, for an internal node in the tree, the encoding is:

```
< question
[encoding of left subtree]
[encoding of right subtree]
>
```

and the encoding for a leaf is:

```
< answer
>
```

Then change the main() method (or its appropriate helper method) so that it asks the user if they would like the program to read knowledge from the le or not. If they answer yes, then call your readFile() method on the Questioner instance. (Otherwise, the program continues with the simple (hardcoded) decision tree.)

Also, when the user decides to quit playing rounds of 20 Questions, the main() method (or its appropriate helper method) should ask the user if they would like the program to save the current knowledge to a le. If they answer yes, then call your writeFile() method on the Questioner instance. (Otherwise, just end the program.)

Hints:

1. With JFileChooser, use the showSaveDialog method in your writeTree method and use the showOpenDialog method in your readTree method. Don't forget to close the file when you are finished writing out the tree!

2. We strongly advise you to use the Scanner class in the readTree method. In particular, if inFile is a Scanner, then

   inFile.hasNext("<")

   determines if the next symbol is an open angle bracket. You can also

   use inFile.nextLine()

   to read the remainder of a line.

The marker will not edit your file in any way, so make sure it will compile and run for the marker.