CSE12 - Spring 2014 HW #7

20 Questions

(100 points)
Due 11:59pm 20 May 2014

In this assignment you will implement a simple game of 20 questions using a binary tree. Some of the inspiration and instructions for this assignment are based on a similar assignment in University of Washington's CSCI 143.

This assignment is an individual assignment. You may ask Professors/TAs/Tutors for some guidance and help, but you can't copy code. You can, of course, discuss the assignment with your classmates, but don't look at or copy each others code or written answers.

The following files are provided for you and can be found on the HW page:

- TQTree.java
- TwentyQuestions.java

You will submit the following file for this assignment:

- **TQTree.java**
- **tree.txt**

Remember to include the standard information in the header comments of your java file (not your tree.txt file!)

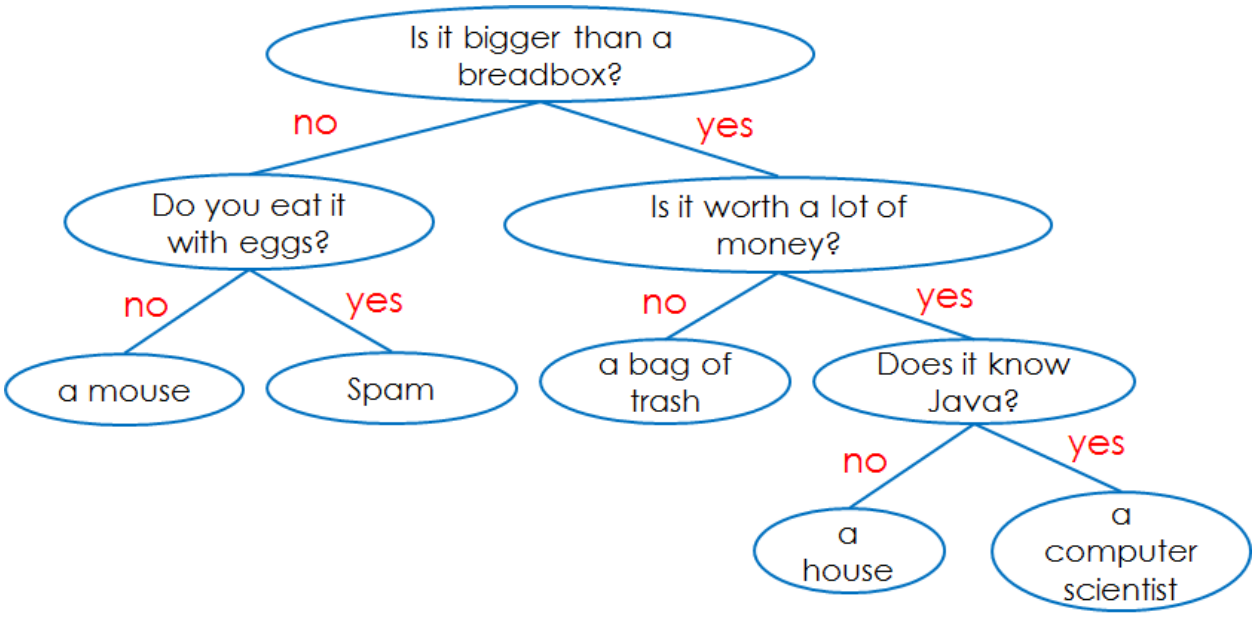
```
CSE 12 Homework 7
Your Name
Your PID
Section [Your section] (A00 for Alvarado, B00 for Papadopoulos)
The date
```

20 Questions Overview

In the game 20 questions the player begins by thinking of an object. The computers goal is to successfully guess what that object is. The computer will ask the player a set of 20 questions (or more or less), attempting to narrow down the list of possible results until it finally thinks it knows the answer. It will then present the result to the player. If the computer successfully guesses the object it wins, otherwise it loses. Note: That our game is simply a set of questions it will likely not ask 20.

To get a feel for the spirit of the game we will implement, you can play an online version of the 20 questions game here: http://www.20q.net/ Of course our game will be simpler in that the user will be restricted to only yes or no questions and answers.

The "knowledge" the computer stores about the world can be represented as a binary tree, as in the figure below. Each non-leaf node stores a question that helps the computer narrow down the space of possible answers, while each leaf node stores the computer's guess.



Here is the same tree in the file format our twenty questions game will support. More detail on this file format is given below. Notice that this file format is a pre-order traversal of the tree:

```
Q:Is it bigger than a breadbox?
Q:Do you eat it with eggs?
A:a mouse
A:spam
Q:Is it worth a lot of money?
A:a bag of trash
Q:Does it know Java?
A:a house
A:a computer scientist
```

Game flow

The game can be run either by loading an existing 20 questions tree from a file (see file format below), or by starting with a default tree (also described below). Then game play proceeds as in the following examples. In these examples the computer's output is shown in black, while the user's responses are shown in blue.

```
> java TwentyQuestions
Is it bigger than a breadbox?
no
Is it spam?
no
OK, what was it?
a mouse
Give me a question that would distinguish a mouse from spam
Do you eat it with eggs?
And would the answer to the question for a mouse be yes or no?
no
Would you like to play again?
yes
Is it bigger than a breadbox?
yes
Is it a computer scientist?
no
OK, what was it?
a bag of trash
Give me a question that would distinguish a bag of trash from a
computer scientist
Is it worth a lot of money?
And would the answer to the question for a bag of trash be yes or no?
no
Would you like to play again?
no
Your final game tree was:
0:  'Is it bigger than a breadbox?'  no:(1)  yes:(2)
1:  'Do you eat it with eggs?'  no:(3)  yes:(4)
2:  'Is it worth a lot of money?'  no:(5)  yes:(6)
3:  'a mouse'  no:(null)  yes:(null)
4:  'spam'  no:(null)  yes:(null)
5:  'a bag of trash'  no:(null)  yes:(null)
6:  'a computer scientist'  no:(null)  yes:(null)
Would you like to save your game?
yes
Please enter a filename to save your game
tree1.txt
Game saved.
Goodbye!

> java TwentyQuestions tree1.txt
```

```
Is it bigger than a breadbox?
yes
Is it worth a lot of money?
yes
Is it a computer scientist?
no
OK, what was it?
a house
Give me a question that would distinguish a house from a computer
scientist
Does it know Java?
And would the answer to the question for a house be yes or no?
no
Would you like to play again?
no
Your final game tree was:
0:  'Is it bigger than a breadbox?'  no:(1)  yes:(2)
1:  'Do you eat it with eggs?'  no:(3)  yes:(4)
2:  'Is it worth a lot of money?'  no:(5)  yes:(6)
3:  'a mouse'  no:(null)  yes:(null)
4:  'spam'  no:(null)  yes:(null)
5:  'a bag of trash'  no:(null)  yes:(null)
6:  'Does it know Java?'  no:(7)  yes:(8)
7:  'a house'  no:(null)  yes:(null)
8:  'a computer scientist'  no:(null)  yes:(null)
Would you like to save your game?
yes
Please enter a filename to save your game
tree2.txt
Game saved.
Goodbye!
```

These examples do not show all of the details of the game's functionality. More detail is given below, and you can (and should) run the game yourself by downloading the .class files in the HW7 folder.

What you have to do

Implement missing functionality

You must implement all of the functionality missing from the starter code to support the game play described above and implemented in the 20 questions application we provide.

Here are the missing methods:

`public TQTree()`

Constructs a new TQTree with one question (“Is it bigger than a breadbox”) and two answers (yes=”spam”; no=”a computer scientist”).

`public TQTree( String filename )`

Constructs a new TQTree by reading from a file containing a twenty questions tree. The format is as described above. Each line in the file is a node in the tree. Question (non-leaf nodes) lines start with “Q:” while answer (leave nodes) lines start with “A:”. Note that “Q:” and “A:” should not be read into the tree. These are there just to help with the file parsing.

Details and hints about the one-argument constructor:

- Notice that this method is already started for you and uses a LineNumberReader to read from a file. You should look at Java’s documentation to understand how to use this class to read from a file.
- If there are any problems reading from the file specified including any kind or IOException or any file format errors, this constructor should print an error message, including the line number where the error occurred if applicable, and then build the default tree described above in the zero-argument constructor. **Errors reading from the file should not cause the Twenty Questions program to crash.**
- You will notice that we have provided a recursive helper method called buildSubtree. It will be extremely useful here. Notice that it throws a ParseException if it encounters any errors while trying to build the subtree from the file. In order to use this method you need to understand what it does, but we also encourage you to try to understand how it works. It’s not necessary to understand it completely… yet. But it will be for HW8, so you might as well get a head start.

`public void play()`

Plays one round of the Twenty Questions game, following the game play outlined above. Namely:

- Starts by asking the user the question in the root node.
- If the user says Yes it goes to the left side, if they say No it goes to the right side.
- When it gets to a leaf node it asks the user if it guessed right.
    - If so it won!
    - If not then it asks the user what object they were thinking of, and
    - A question that differentiates their object from the one i tree guessed, and
    - Whether their answer to that question is yes or no
- It then uses this information to augment the tree.

Check out the sample executions above to see how this will look.

Details and hints about the play method:

- As you play one round, your method walks down the tree, choosing to go to the yes child or the no child based on whether the user answers yes or no to each question. You will need to determine a way to determine when your method should make a "guess" rather than asking another question.
- If the computer does not guess the user's answer, this method needs to augment the tree with a new question and a new object. This will involve inserting two new nodes into the tree. We recommend you draw several examples on paper to see how you'll handle modifying your tree before you start coding. Figure out what nodes you need references to, and how their pointers will change based on the information the user gives you *before* you start trying to code this.
- The computer should accept any of "Y", "y", "yes", "Yes", or "YES" as affirmative answers. Leading and trailing whitespace should be ignored (e.g., " yes " is the same as "yes"). Non-affirmative answers should be treated as negative answers.
- It's fine if the computer saves text into the question tree exactly as the user enters it. E.g. if the user enters "A MOUSE" it's OK to simply leave the text in all caps. However, you are also welcome to do any text processing you like to make the text stored in the game tree more standard. A couple ideas include:
    - Standardizing capitalization
    - Adding question marks to the end of questions

```
public void save( String filename ) throws FileNotFoundException
```
Saves the tree in the format specified above to the file given. If the file cannot be created or written to, it throws a FileNotFoundException.

```
public void print()
```
Prints a level-order traversal of the tree to standard output. Each node in the printed tree should be labeled with a number in order, and each node should print the numbers of its left and right children.

Print will be called by the main method provided to display the final tree to the user before asking whether the user wants to save it or not, as in the examples above.

Note: The key to this method is to do two breadth first searches of your tree to do the two level-order traversals. In the first you will mark each node with its index (in level-order) and in the second you will print the nodes (with information about their children). Last time you implemented BFS you used your custom queue. For this method, it's easy enough to simply use a LinkedList as a queue (i.e. add from one end and remove from the other).

Remember to keep your methods short and write helper methods where appropriate.

Generate an interesting game tree

Once you have your game play working, play your 20 questions game to generate an interesting game tree. Save this tree in a file called tree.txt.

Grading

Points will be awarded on this HW as follows:

- 5 points: Zero-argument constructor
- 10 points: 1-argument constructor
- 25 points: play
- 20 points: save
- 20 points: print
- 5 points: Your submitted game tree
- 15 points: Documentation and style