MSCI 240, Fall 2018, Project 3
Assigned: November 16, 2018
Due: December 3, 2018, 5:00pm

Instructions:

Please submit all answers for this assignment **printed** on paper to the drop box located on the **third** floor between E2 and CPH AND, any **".java" files** and **".pdf" files** to the Project 3 dropbox on LEARN (specific instructions are provided below).

Submission: You should save the program using the original filenames of the code, and **upload them to the dropbox** on Learn with all of your changes.

* Your code printout should be printed **straight from the Eclipse IDE**. Else, it must be neatly formatted and look exactly as it does in the Eclipse IDE. Failure to follow instructions will result in penalty.

You must create a cover page that includes the following information:

1. Your name.
2. The course info: "MSCI 240 Fall 2018, Instructor: Dr. Mark Hancock"
3. The date on which you are submitting your assignment.
4. Which assignment this is.
5. A completed and hand signed MSCI 240 acknowledgements page. A blank statement is in UW-Learn in Content->Course Material. **Remember your signature!**
6. All pages must be letter-sized pieces of paper. **You must staple** multiple pages together with one metal staple in the upper left hand corner.

Failure to meet requirements will result in a significant loss of points.

Collaboration and Academic Honesty

1. Please review the syllabus for guidelines and extensive rules on academic honesty in this course.
2. You are to work alone on this assignment except that:
    a. You may talk with the instructor and the TA regarding this homework.
    b. You may talk with students about general programming questions, but you may not discuss ways to solve this homework. For example, you may have another student help explain loops or conditionals or variables to you, but you may not discuss how to write parts of a program to solve this homework.
3. All collaboration other than that allowed above (2a, 2b) is forbidden. For example:
    a. You may not discuss the problems with people.
    b. You may not sketch out pseudo-code on a whiteboard.
    c. You may not share code with others.
    d. You may not give answers to others.
    e. You may not compare answers with others.
4. In addition, you may not use any materials other than your textbook, books you can find in the library, and Java API documentation. You must cite any source that you use.
    a. **This means that you may not search for answers on the internet.**
5. We will use automatic methods to detect cheating as well as human inspection of solutions.
6. Violating these rules will be considered academic misconduct and subject to penalties.

This project is based on textbook Programming Project 17.1 (written by Stuart Reges & Marty Stepp).
This program focuses on *binary trees* and *recursion*. You will need support files `UserInterface.java`, `QuestionMain.java`, and input text files from Learn; place them in the same folder as your class. Turn in files `QuestionTree.java` and `QuestionNode.java`.

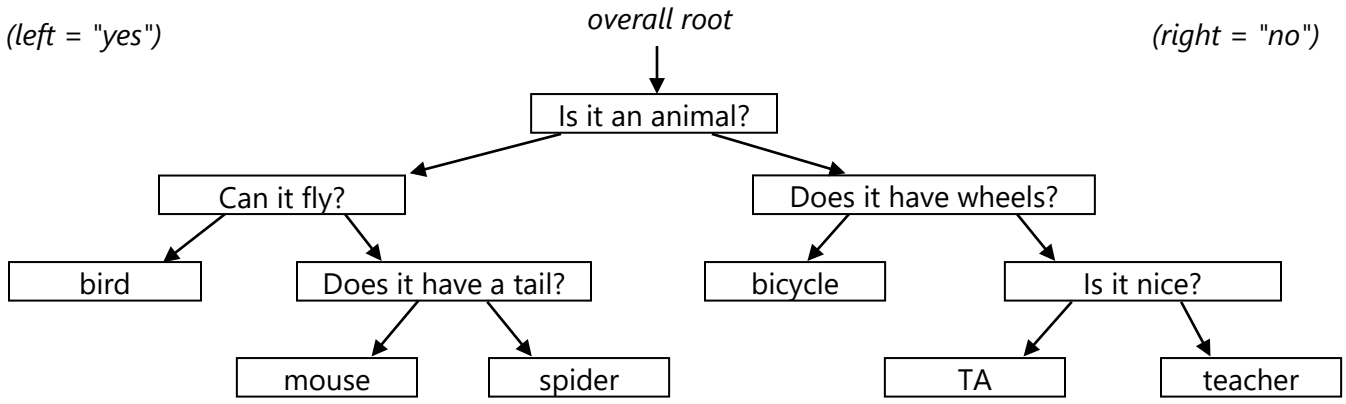
The Game of 20 Questions

In this assignment, you will implement a yes/no guessing game called "20 Questions". Each round of the game begins by you (the human player) thinking of an object. The computer will try to guess your object by asking you a series of yes or no questions. Eventually the computer will have asked enough questions that it thinks it knows what object you are thinking of. It will make a guess about what your object is. If this guess is correct, the computer wins; if not, you win.

The computer 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.) A "question" node contains a left "yes" subtree and a right "no" subtree. An "answer" node is a leaf. The idea is that this tree can be traversed to ask the human player a series of questions.

For example, in the tree below, the computer begins the game by asking, "Is it an animal?" If the player says "yes", the computer goes left to the "yes" subtree and then asks the user, "Can it fly?" If the user had instead said "no", the computer would go right to the "no" subtree and then ask the user, "Does it have wheels?"

This pattern continues until the game reaches a leaf "answer" node. Upon reaching an answer node, the computer asks whether that answer is the correct answer. If so, the computer wins.



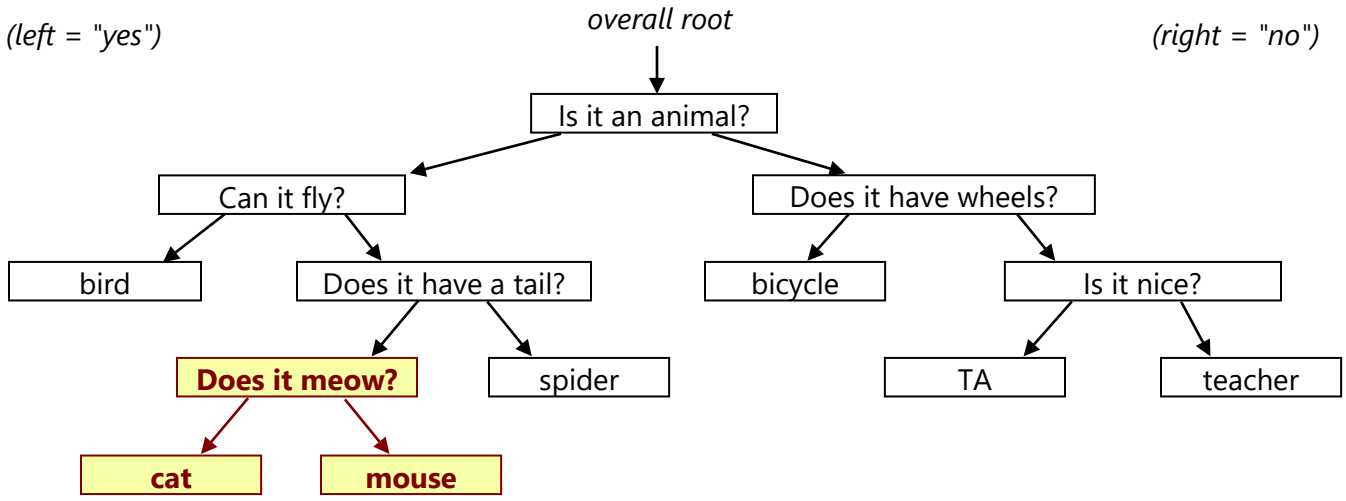
The following partial output log shows one game being played on the above tree:

```
Is it an animal? yes
Can it fly? no
Does it have a tail? yes
Would your object happen to be mouse? yes
I win!
```

Initially the computer is not very intelligent, but grows more intelligent each time it loses. If the computer's answer guess is incorrect, the player gives it a new question it can ask to help it in future games. For example, suppose in the preceding log that the player was thinking of cat rather than a mouse. The game log might be:

```
Is it an animal? yes
Can it fly? no
Does it have a tail? yes
Would your object happen to be mouse? no
I lose. What is your object? cat
Type a yes/no question to distinguish your item from mouse: Does it meow?
And what is the answer for your object? yes
```

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. For example, after the game represented by the preceding log, the computer's overall game tree would be:



In this assignment, you will create classes `QuestionTree` and `QuestionNode` to represent the computer's tree of yes/no questions and answers for playing games of 20 Questions. You are provided with a client `QuestionMain` that handles user interaction and calls your tree's methods to play games. Below are two logs of execution (user input is underlined):

Log of execution #1

```
Welcome to the game of 20 Questions!
Shall I recall our previous games? n

Think of an item, and I will guess it.

Would your object happen to be Jedi? n
I lose. What is your object? hobbit
Type a yes/no question to distinguish your
item from Jedi: Does it use the Force?
And what is the answer for your object? n
Challenge me again? y

Does it use the Force? n
Would your object happen to be hobbit? n
I lose. What is your object? droid
Type a yes/no question to distinguish your↵
item from hobbit: Is it metal?
And what is the answer for your object? y
Challenge me again? y

Does it use the Force? n
Is it metal? y
Would your object happen to be droid? y
I win!
Challenge me again? n

Games played: 3
I won: 1
Shall I remember these games? y
What is the file name? question1.txt
```

Log of execution #2

```
Welcome to the game of 20 Questions!
Shall I recall our previous games? yes
What is the file name? question2.txt

Think of an item, and I will guess it.

Is it an animal? no
Does it have wheels? yes
Would your object happen to be bicycle? yes
I win!
Challenge me again? yes

Is it an animal? yes
Can it fly? no
Does it have a tail? yes
Would your object happen to be mouse? no
I lose. What is your object? cat
Type a yes/no question to distinguish your↵
item from mouse: Does it meow?
And what is the answer for your object? yes
Challenge me again? yes

Is it an animal? yes
Can it fly? no
Does it have a tail? yes
Does it meow? yes
Would your object happen to be cat? yes
I win!
Challenge me again? no

Games played: 3
I won: 2
Shall I remember these games? no
```

The program can instruct your question tree to read its input from different text files. You should initially test with the provided `question1.txt` or an even smaller file of your own creation. As your code runs, you will be able to create larger files by saving them at the end of the program. Once your code works with `question1.txt`, you can test it with a larger input such as `animals.txt`, which comes with permission from the Animal Game web site at http://animalgame.com/.

Implementation Details:

The contents of the `QuestionNode` class are up to you. Though we have studied trees of `int`s, in this assignment you can create nodes specific to solving this problem. Your node class should have at least one constructor used by your tree. Your node's fields can be `public`. `QuestionNode` should not perform a large share of the overall game algorithm.

Your `QuestionTree` must have the following members. You may add extra private methods. You should **throw an `IllegalArgumentException`** if the parameter passed to any method below is `null`.

`public QuestionTree(Scanner input, PrintStream output)`

In this constructor you should initialize your new question tree. You are passed objects representing the user interface for input/output (typically from `System.in` and `System.out`). Your tree will use this user interface for printing output messages and asking questions in the game. Initially the tree starts out containing only a single answer leaf node with the word "Jedi" in it. The tree will grow larger as games are played or as a new tree is loaded with the `load` method described below.

`public void play()`

A call to this method should 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, print a message saying so. Otherwise your tree must ask the user what object he/she was 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. The two boxed partial logs on the first page are examples of output from single calls to `play`. All user input/output should be done through the `Scanner` and `PrintStream` objects passed to your tree's constructor.

After the game is over, the provided client program will prompt the user whether or not to play again; this is not part of your play method. Leave this functionality to the client program.

`public void save(PrintStream output)`

In this method you should store the current tree state to an output file represented by the given `PrintStream` (note: this is not the same `PrintStream` as the one passed to the constructor—it's from a file, not `System.out`). In this way your question tree can grow each time the user runs the program. (You don't save the number of games played/won.) A tree is specified by a sequence of lines, one for each node. Each line must start with either Q: to indicate a question node or A: to indicate an answer (a leaf). All characters after these first two should represent the text for that node (the question or answer). The nodes should appear in the order produced by a *preorder traversal* of the tree. For example, the two trees shown in the diagram and logs on the preceding pages would be represented by the following contents:

question1.txt

```
Q:Does it use the Force?
A:Jedi
Q:Is it metal?
A:droid
A:hobbit
```

question2.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
```

`public void load(Scanner input)`

In this method you should replace the current tree by reading another tree from a file. Your method will be passed a `Scanner` that reads from a file (note: this is not the same `Scanner` as the one passed to the constructor—it's from a file, not `System.in`) and should replace the current tree nodes with a new tree using the information in the file. Assume the file exists and is in proper standard format. Read entire lines of input using calls on `Scanner`'s `nextLine`. (You don't load the number of games played/won, just the tree. Calling this method doesn't change games played/won.)

`public int totalGames()`
`public int gamesWon()`

In these methods you should respectively return the total number of games that have been played on this tree so far, and the number of games the computer has won by correctly guessing your object, during the current execution of the program. Initially 0 games have been played and won, but the games played increase by 1 for each game that is played (each time `play` is called), and the games won increase by 1 each time the computer guesses your object correctly.

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 `save` method. For full credit, this must be in proper format, have a height of at least 5, and be your own work.

Style Guidelines and Grading:

Part of your grade will come from appropriately using binary trees and recursion to implement your guessing game 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.

Redundancy is another major grading focus; some methods are similar in behaviour or based off each other's behaviour. You should avoid repeated logic as much as possible. Your class may have other methods besides those specified, but any other methods you add should be `private`.

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. 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. This is considered bad style. 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.

You should follow good general style guidelines such as: making fields `private` and avoiding unnecessary fields (you will lose points if you declare variables as fields that could instead be declared as local variables); appropriately using control structures like loops and `if/else`; properly using indentation, good variable names and types; and not having any lines of code longer than 100 characters.

Comment your code descriptively in your own words at the top of your class, each method, and on complex sections of your code. Comments should explain each method's behaviour, parameters, return, pre/post-conditions, and exceptions. For reference, our solution is around 130 lines long (67 "substantive lines"), including comments and blank lines.