Homework Turnin

Name:	Akshit K Patel
Email:	akshit@uw.edu
Student ID:	1561387
Section:	DC
Course:	CSE 143 16au
Assignment:	a7
Receipt ID:	a1f03c9d28de3ba045b3d1ec7ea4a138

Warning: Your turnin is 1 day late. Assignment a7 was due Thursday, December 1, 2016, 11:30 PM.

Turnin Successful!

The following file(s) were received:

QuestionsGame.java (13079 bytes)

```java
/**
 * @author Akshit Patel
 * @Date 11/24/2016
 * CSE 143D DC
 * TA: Melissa Medsker
 * HW #7 20 Questions
 */
import java.io.PrintStream;
import java.util.Scanner;

/**
 * QuestionsGame represents a game of N-questions where the computer plays with
 * user by guessing the answer to the questions given in a standard format text
 * file. The class also updates the questions with its associated correct answer
 * when the answer guessed is wrong. The class also provides a useful method to
 * store the updated session of new questions and answers in standard form as a
 * text document, overwriting the original given text document.
 *
 * <p>
 * Standard format of text document(.txt) considered for this class follows the
 * rules:
 * <ul>
 *  <li>The first line of document is either question(Q:) and then the
 *  associated question or an answer(A:) and then the associated answer.
 *  Like: <br>
 *  <ul>
 *      <i>
 *          Q:<br>
 *          is it an aninal?<br>
 *          A:<br>
 *          Dog<br>
 *      </i>
 *  </ul>
 *  <li>Every question has to have a non-empty sequence of line pairs. i.e. it
 *  cannot be:<br>
 *  <ul>
 *      <i>
 *          Q:<br>
 *          is it an aninal?<br>
 *          A:<br>
 *          Dog<br>
 *      </i>
 *  </ul>
 *  but has to be more like:<br>
 *  <ul>
 *      <i>
 *          Q:<br>
 *          is it an aninal?<br>
 *          A:<br>
 *          Dog<br>
 *          A:<br>
 *          Human<br>
 *      </i>
 *  </ul>
 *  where there is answer to question for yes or no. NOTE: there could be
 *  another linked question instead of answer but it has to follow the same rule
 * </ul>
 * </p>
 *
 */
public class QuestionsGame {

    /*
     * Overall root of the question tree.
     */
    private QuestionNode root;

    /**
     * Constructs a new QuestionGame object representing the one given string
     * object.
     *
     * @param object String representation of the object to be considered for
     *        this QuestionGame. The String cannot be null
     */
    public QuestionsGame(String object) {
        this.root = new QuestionNode(object);
    }

    /**
     * Constructs a new QuestionGame object from a given scanner containing the
     * questions and answers in standard format.
     *
     * @param input Scanner containing questions and answers. The given scanner
     *        is not null and is attached to a legal, existing file in Standard
     *        format
     */
    public QuestionsGame(Scanner input) {
        this.root = this.getQuestions(input);
    }

    /**
     * Constructs a new QuestionGame question tree from a given scanner
     * containing the questions and answers in standard format.
     *
     * @param input Scanner containing questions with answers in standard
     *        format. The given scanner is not null and is attached to a legal,
     *        existing file in Standard format
     * @return QuestionNode of the question tree made for this QuestionGame
     *         object.
     */
    private QuestionNode getQuestions(Scanner input) {
        QuestionNode current = null;
        // make a branch if scanner has elements left to consider.
        if (input.hasNextLine()) {
            // get the type, either Q: or A:
            String type = input.nextLine();
            // get the actual answer or question.
            String data = input.nextLine();
            // if answer then we have a leaf.
            if (type.equals("A:")) {
                return new QuestionNode(data);
            }
            // otherwise construct a new branch to continue building.
            current = new QuestionNode(data);
            // construct the left and right branch.
            current.left = this.getQuestions(input);
            current.right = this.getQuestions(input);
        }
        // return the root of the question tree formed.
        return current;
    }

    /**
     * Stores the current questions and answers to an output file represented by
     * the given PrintStream. This method is useful to store the question and
     * answer when incorrect guesses are made as new questions and answers are
     * added and thus can be used to later play another game with computer using
     * updated file. The file is made in standard format and overwrites data of
     * the given text document.
     *
     * @param output PrintStream representing the text file to store the current
     *        question and answers of this QuestionGame object in standard
     *        format.
     * @throws IllegalArgumentException if the given PrintStream is null.
     */
    public void saveQuestions(PrintStream output) {
        if (output == null) {
            throw new IllegalArgumentException("File cannot be null!");
        }
        this.readTree(output, this.root);
    }

    /**
     * Reads the question tree considered for this QuestionGame object and
     * stores it in standard format to a output file represented by the given
     * PrintStream.
     *
     * @param output PrintStream representing the text file to store the current
     *        question tree of this QuestionGame object in standard format. It
     *        should not be null.
     * @param current QuestionNode of the question tree considered for this
     *        QuestionGame object. Initially the root(not null). Used to read
     *        and store the question tree in pre-order (Standard format)
     */
    private void readTree(PrintStream output, QuestionNode current) {
        // store the Answer if its a leaf.
        if (current.left == null && current.right == null) {
            output.println("A:");
            output.println(current.data);
        } else {
            output.println("Q:");
            output.println(current.data);
            // store the remaining left and right branches.
            this.readTree(output, current.left);
            this.readTree(output, current.right);
        }
    }

    /**
     * This method plays one complete guessing game with the user by using the
     * current question tree to ask questions and eventually guesses the answer
     * based on user reply (handled by the method). Computer prints a message
     * saying that it won if the guess made is correct, otherwise it asks the
     * user the following questions:<br>
     * <ul>
     *  <li>what object they were thinking of,
     *  <li>a question to distinguish that object from the player guess, and
     *  <li>whether the player object is the yes or no answer for that question.
     * </ul>
     * thus adding new questions and answers to this QuestionGame object.
     *
     * <p>
     * If a user reply is any word beginning with letter <b>y or Y</b>, it is
     * considered to be a yes reply and any other beginning considered to be a
     * no.
     * </p>
     */
    public void play() {
        // scanner to get user input.
        Scanner getAns = new Scanner(System.in);
        // play the game and update the tree if needed.
        this.root = this.getAnswer(this.root, getAns);
    }

    /**
     * Plays one complete guessing game with the user by using the current
     * question tree to ask questions and eventually guesses the answer based on
```

```java
     * user reply (given a scanner). Computer prints a message saying that it
     * won if the guess made is correct, otherwise it asks the user the
     * questions as described in method {@link play} in order to update the
     * current question tree of this QuestionGame object to the new by getting
     * the correct guess object and it associated question. Only the incorrect
     * branch of the tree is changed.
     *
     * @param current QuestionNode of the question tree considered for this
     *        QuestionGame object. Initially the root(not null). Used to read
     *        and modify the question tree in pre-order (Standard format).
     * @param input Scanner representing the user reply, it cannot be null.
     * @return QuestionNode of the question tree read or modified for this
     *         QuestionGame object.
     */
    private QuestionNode getAnswer(QuestionNode current, Scanner input) {
        // if we have a leaf, we have a possible answer.
        if (current.left == null && current.right == null) {
            System.out.println("I guess that your object is " + current.data
                               + "!");
            System.out.print("Am I right? (y/n)? ");
            // if user input start with y, then computer wins.
            if (input.nextLine().trim().toLowerCase().startsWith("y")) {
                System.out.println("Awesome! I win!");
            } else {
                // reference the current node to the new node.
                current = this.updateTree(input, current);
            }
        } else {
            // print the question, ask for response
            System.out.print(current.data + " (y/n)? ");
            // if response is yes, go read/modify left.
            if (input.nextLine().trim().toLowerCase().startsWith("y")) {
                current.left = getAnswer(current.left, input);
            } else {
                current.right = getAnswer(current.right, input);
            }
        }
        // return the read/modified tree for this QuestionGame object.
        return current;
    }

    /**
     * Updates the current question tree with new question to be added with its
     * associated answers and also handles the interaction with the player as
     * mentioned in {@link getAnswer} to ask for the questions and answers if
     * given with a scanner.
     *
     * @param input Scanner representing the user reply, it cannot be null
     * @param current QuestionNode of the question tree considered for this
     *        QuestionGame object. Initially the leaf where the question needs
     *        to be added. Used to read and modify the question tree.
     * @return new QuestionNode of the modified question tree.
     */
    private QuestionNode updateTree(Scanner input, QuestionNode current) {
        System.out.println("Boo! I Lose." + "  Please help me get better!");
        System.out.print("What is your object? ");
        // get the user object.
        String object = input.nextLine();
        System.out.println("Please give me a yes/no question that "
                           + "distinguishes between "
                           + object
                           + " and "
                           + current.data
                           + ".");
        System.out.print("Q: ");
        // get the user defined question.
        String question = input.nextLine();
        System.out.print("Is the answer \"yes\" for " + object + "? (y/n)? ");
        // if user response is yes, the user object is at left node.
        if (input.nextLine().trim().toLowerCase().startsWith("y")) {
            return new QuestionNode(question, new QuestionNode(object),
                    current);
        }
        // otherwise, its the right node.
        return new QuestionNode(question, current, new QuestionNode(object));
    }

    /**
     * QuestionNode creates a simple binary tree of nodes with string data
     */
    private static class QuestionNode {
        /**
         * Data to be stored in the node, it cannot be changed.
         */
        public final String data;
        /**
         * Representing the left node of the binary tree, used to store answers
         * and questions in QuestionGame object question tree.
         */
        public QuestionNode left;
        /**
         * Representing the right node of the binary tree, used to store answers
         * and questions in QuestionGame object question tree.
         */
        public QuestionNode right;

        /**
         * Constructs a new binary tree leaf with given data.
         *
         * @param data String representation of the data to be stored in the
         *        nodes of the binary tree. Should not be null
         */
        public QuestionNode(String data) {
            this(data, null, null);
        }

        /**
         * Constructs a new binary tree with given data and its left & right
         * QuestionNode.
         *
         * @param data String representation of the data to be stored in the
         *        nodes of the binary tree. Should not be null
         * @param left representing the left node of the binary tree
         * @param right representing the right node of the binary tree
         */
        public QuestionNode(String data,
                            QuestionNode left,
                            QuestionNode right) {
            this.data = data;
            this.left = left;
            this.right = right;
        }
    }
}
```

myquestions.txt (663 bytes)

```
Q:
Is the person an Soccer Athlete?
Q:
Is the person the best player?
A:
lionel messi
Q:
Is it sencond best?
A:
Cristiano Ronaldo
Q:
does he play in England?
Q:
does he play for Manchester United?
Q:
is he a striker?
A:
Zlatan Ibrahimovic
Q:
is he a midfielder?
Q:
does he dab?
A:
Paul Pogba
A:
Juan Mata
A:
David De Gea
Q:
Does he play for Chelsea?
```

```
A:
Eden Hazard
A:
Kevin De Bruyne
A:
Luis Suarez
Q:
Is the person a cricketer?
Q:
is he the best?
A:
Virat Kohli
Q:
Is he indian?
A:
MS Dhoni
A:
Chris Gayle
Q:
Is the person a common man?
A:
Akshit Patel
Q:
Is the person a celebrity?
A:
Tom cruise
A:
God
```