Assignment 5 – Guessing Game – Cathal Lawlor 21325456

Problem Statement

Being able to employ the code we have used in class before, I will have to develop a method of saving the objects (binary nodes) to storage, using serialising and deserializing, these objects can be saved and loaded from memory.

There is a sample base tree created for the user to play.

The game is called the Expandable Binary Tree Guessing Game, which involves a series of questions asked to the user to guess an object, thing or concept. Initially, there is a pre-defined set of questions and guesses in a binary tree. If the program's guess is incorrect, the user is prompted to provide the correct answer and a new question to distinguish the guess from the actual answer. The program then updates the binary tree with the new question and its corresponding yes/no answers. Additionally, the user can save and load the binary tree from a file for future use.

Analysis and Design

To solve this problem, we will use a BinaryNodeInterface and an expandable binary tree. The initial binary tree will be manually built with at least four levels, where each internal node is a yes/no question and each leaf node is a guess. The program will traverse the tree using the user's answers to the questions until it reaches a leaf node, at which point it will either provide the correct guess or ask the user for the correct answer and a new question to differentiate the guess from the actual answer. The program will then replace the leaf node with the new question and its yes/no answers.

The following are the methods I'll have and what they'll do:

Constructor - Create an initial binary tree using a createTree method.

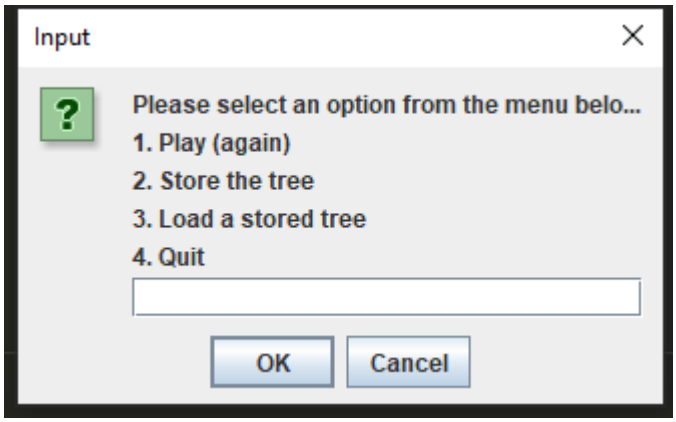
1. InteractiveQuestions - While the user wants to play:
    a. Set the currentNode to the root node of the tree.
    b. While the currentNode is not a leaf node:
        i. Ask the question associated with the currentNode.
        ii. If the answer is yes, set the currentNode to its left child.
        iii. If the answer is no, set the currentNode to its right child.
    c. If the currentNode is a leaf node:
        i. Present the guess to the user.
        ii. If the guess is correct, display options for the user to continue.
            1. Play again?
            2. Store the tree?
            3. Load a stored tree?
            4. Quit?
        iii. If the guess is incorrect, ask the user for the correct answer and a new question to differentiate the guess from the actual answer.
        iv. Replace the currentNode with the new question and its yes/no answers.
2. treeIO.save - Save the binary tree in a file using a suitable method, such as a serialized object or a specifically formatted text file.
3. treeIO.load Load a stored binary tree from a file.
4. Displaytree - Print out a text representation of the contents of the binary tree as breath first.

Loading and saving tree

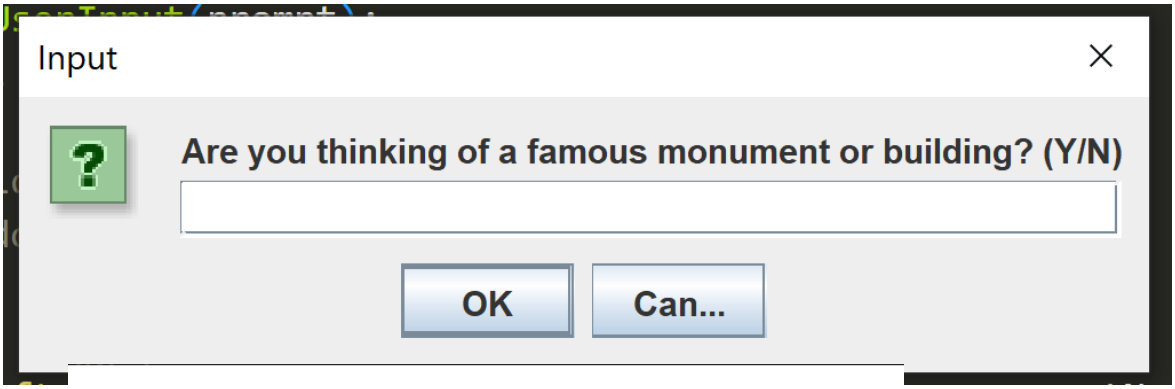
The treeIO class houses the methods for how I serialize and deserialize binary tree objects.
storeTree method takes a binary tree object and saves it to a file named "20QuestionTree.txt". It uses Java's Object Output Stream. loadTree method loads the binary tree object from the "20QuestionTree.txt" file again using the Java Object Input Stream and returns the loaded tree. If the file is not found or there is a class loading exception, it will exit the program.

Testing:

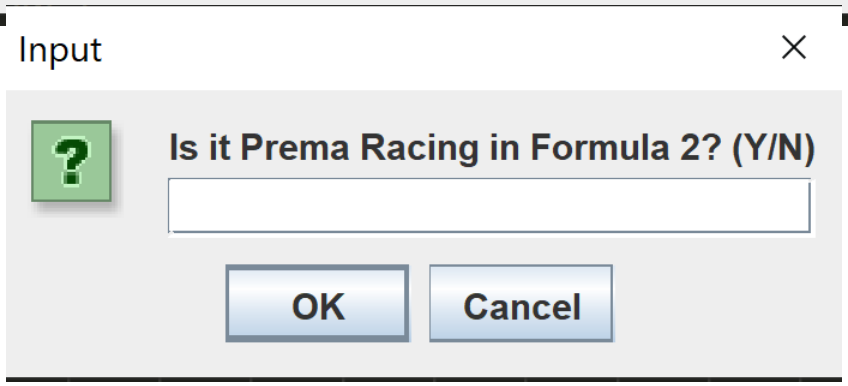
Menu:



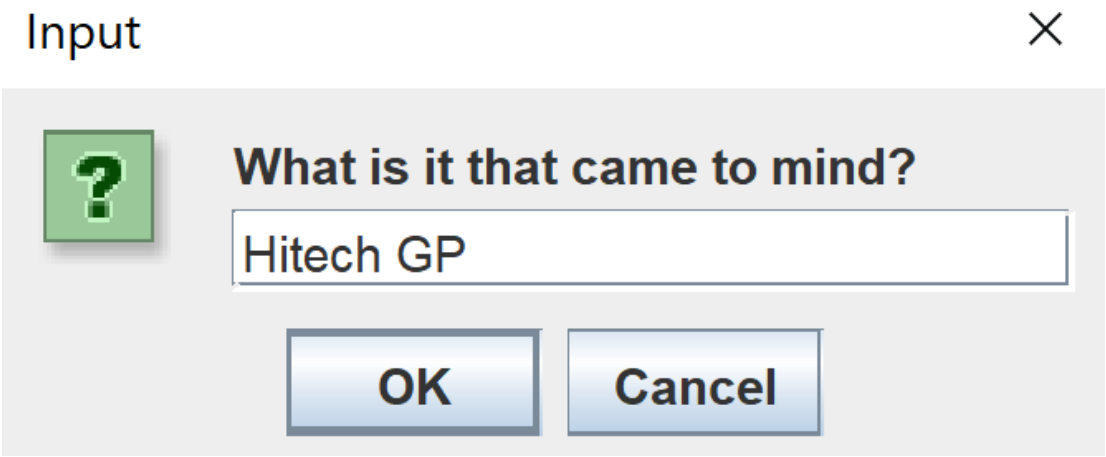
Asking a question:



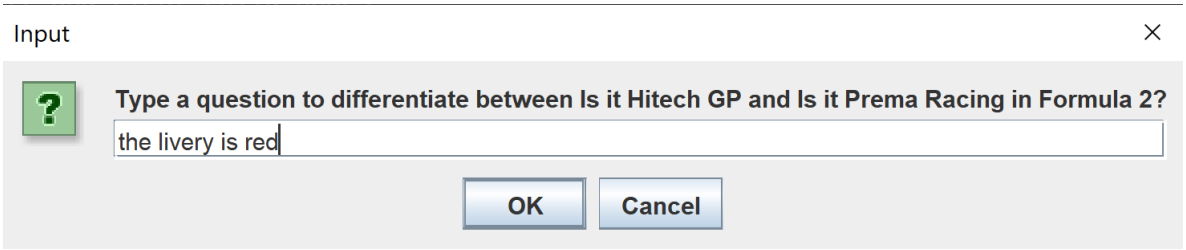
A guess:



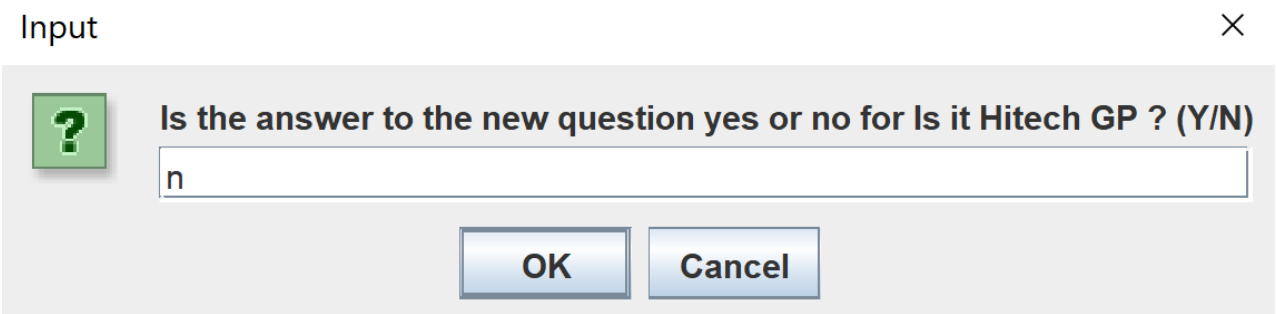
When it is no, it prompt the user for what their guess would be



Asking the user for a new question to differentiate the two nodes

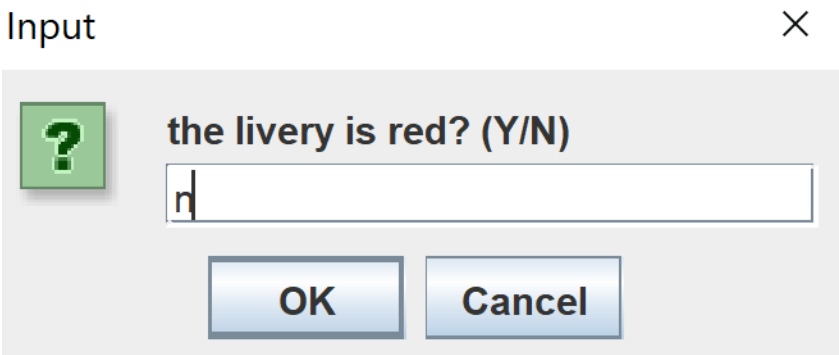


Asking the user is the new node, e.g. Hitech GP the yes or no answer to the new question

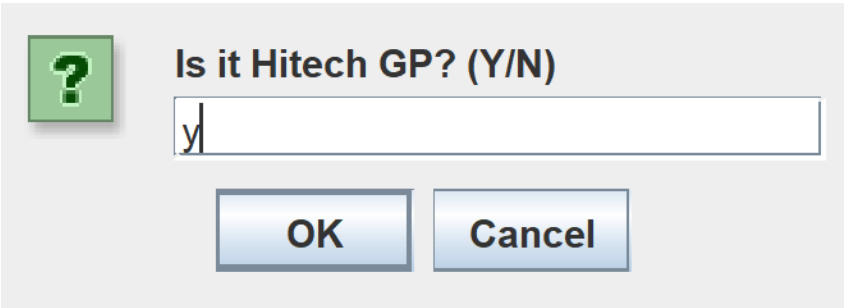


Trying the game again:

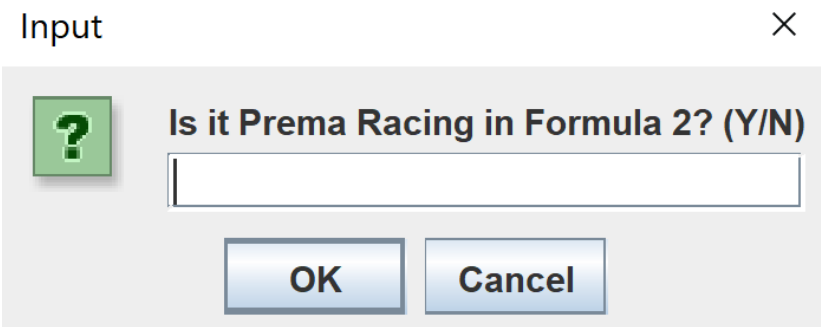
The new input



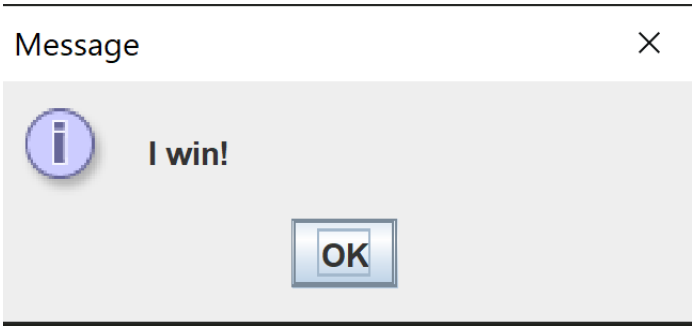
The new guess

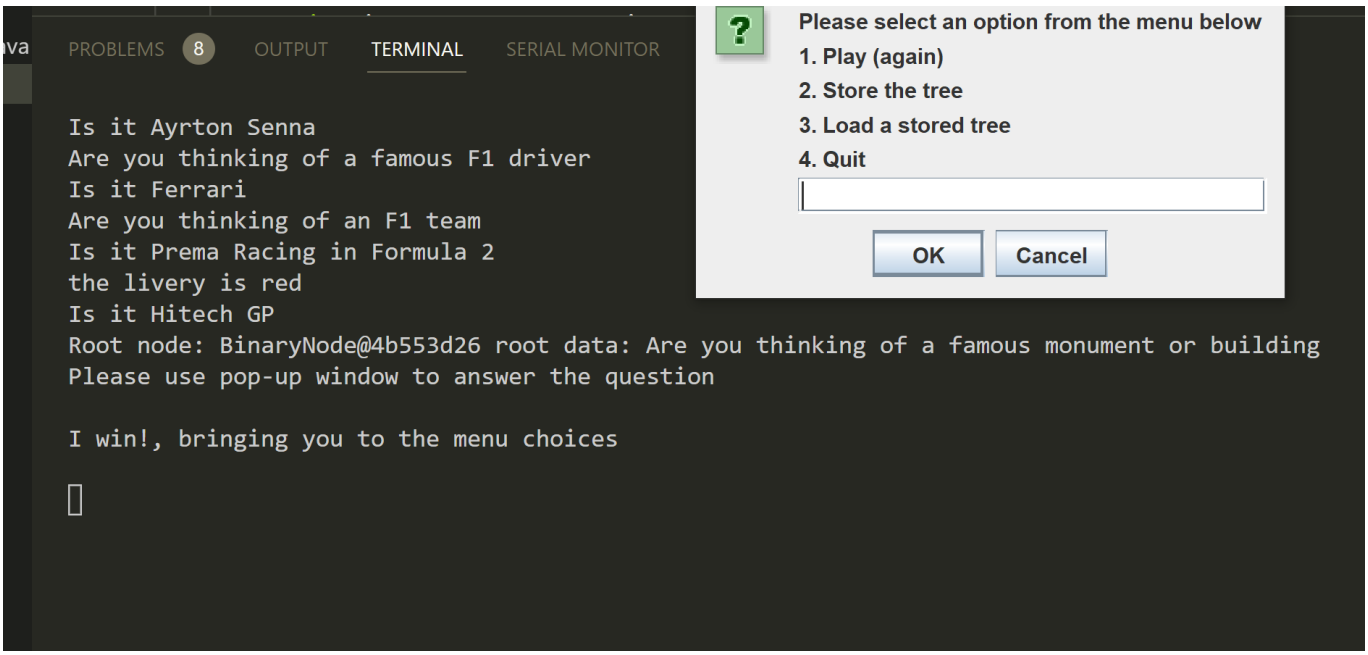


The moved prema guess from before



When the guess is correct, it goes back to menu

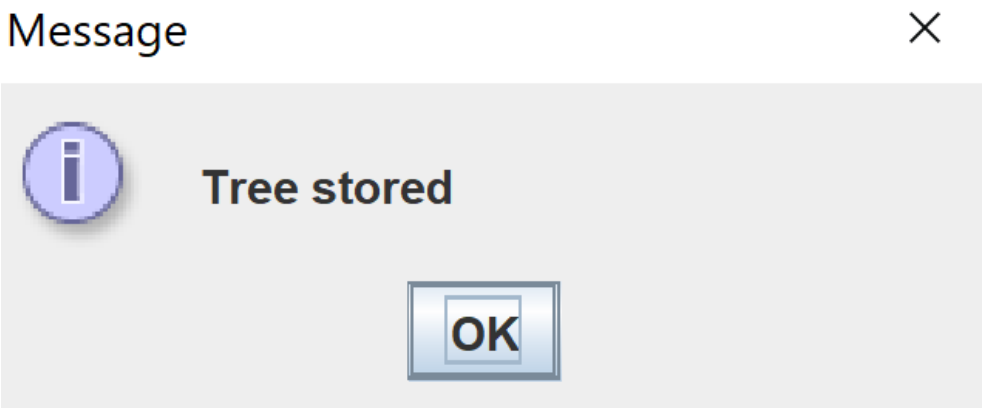




Displaying the tree

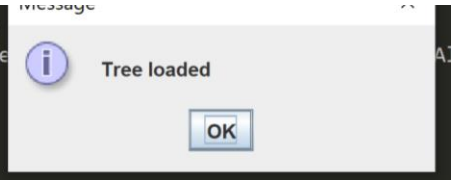
```
 Tree represenation in text:
Node 1: Are you thinking of a famous monument or building
Node 2: Is it in the northern hemisphere
Node 3: Are you thinking of a famous F1 driver
Node 4: Is it in Europe
Node 5: Is it in Asia
Node 6: Are they from Europe
Node 7: Are you thinking of an F1 team
Node 8: Is it the leaning tower of Pisa
Node 9: Is it the Statue of Liberty
Node 10: Is it the Taipei 101
Node 11: Is it Christ the Redeemer in Brazil
Node 12: Is it Michael Schumacher
Node 13: Is it Ayrton Senna
Node 14: Is it Ferrari
Node 15: Is it Prema Racing in Formula 2
```

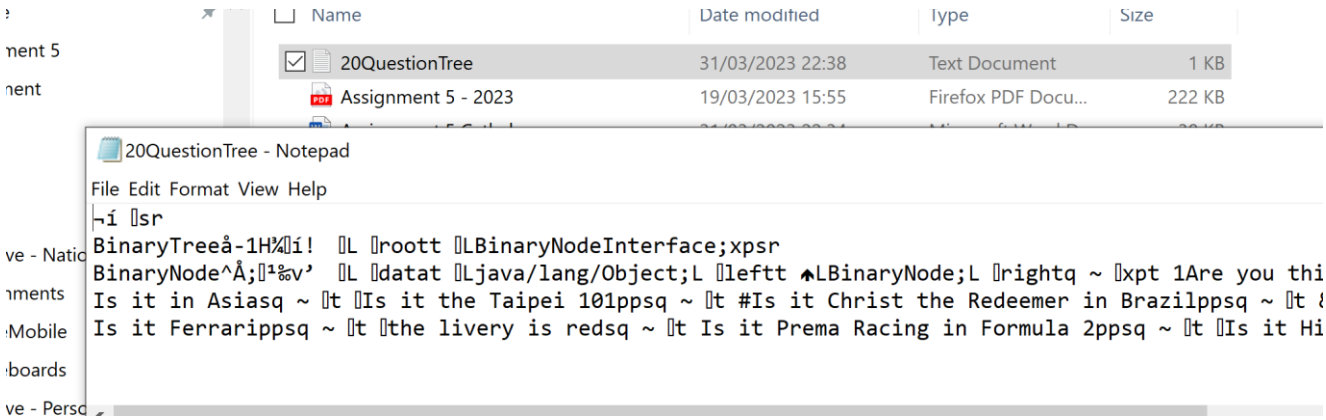
Storing tree



Loading tree

```
The left node: Is it Prema Racing in Formula 2 right
File path: C:\Users\catha\Documents\GitHub\College\Ye
Saving tree
Loading tree from file
Tree:
Is it the leaning tower of Pisa
Is it in Europe
Is it the Statue of Liberty
Is it in the northern hemisphere
Is it the Taipei 101
Is it in Asia
Is it Christ the Redeemer in Brazil
Are you thinking of a famous monument or building
Is it Michael Schumacher
Are they from Europe
Is it Ayrton Senna
Are you thinking of a famous F1 driver
Is it Ferrari
Are you thinking of an F1 team
Is it Prema Racing in Formula 2
the livery is red
Is it Hitech GP
```





Code:

GuessingGame.java

```
//import java.io.File;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

import javax.swing.JOptionPane;

public class GuessingGame {
    static GuessingGame userSession = new GuessingGame();
    static treeIO treeIO = new treeIO();
    String treeStr = "";

    public static void main(String[] args) {
        System.out.println("Creating a base tree ...");
        BinaryTree<String> testTree = new BinaryTree<String>();

        createTree2(testTree);
        BinaryNodeInterface<String> currentNode = testTree.getRootNode();

        // Display some statistics about it
        System.out.println("\nSome statistics about the test tree ...");
        displayStats(testTree);

        // Perform in-order traversal
        System.out.println("\nIn-order traversal of the tree, printing each node when
visiting it ...");

        System.out.println("\n Tree represenation in text: ");
        displayTree(testTree);

        userSession.menuChoice(testTree, currentNode);

    }

    public static void createTree2(BinaryTree<String> tree) {
        // To create a tree, build it up from the bottom:
        // create subtree for each leaf, then create subtrees linking them,
        // until we reach the root.


        BinaryTree<String> oTree = new BinaryTree<String>("Is it Prema Racing in Formula
2");
        BinaryTree<String> nTree = new BinaryTree<String>("Is it Ferrari");
        BinaryTree<String> mTree = new BinaryTree<String>("Is it Ayrton Senna");
        BinaryTree<String> lTree = new BinaryTree<String>("Is it Michael Schumacher");
        BinaryTree<String> kTree = new BinaryTree<String>("Is it Christ the Redeemer in
Brazil");
        BinaryTree<String> jTree = new BinaryTree<String>("Is it the Taipei 101");
        BinaryTree<String> iTree = new BinaryTree<String>("Is it the Statue of Liberty");
```

```java
        BinaryTree<String> hTree = new BinaryTree<String>("Is it the leaning tower of
Pisa");

        // Now the subtrees joining leaves:
        BinaryTree<String> gTree = new BinaryTree<String>("Are you thinking of an F1
team", nTree, oTree);
        BinaryTree<String> fTree = new BinaryTree<String>("Are they from Europe", lTree,
mTree);
        BinaryTree<String> eTree = new BinaryTree<String>("Is it in Asia", jTree, kTree);
        BinaryTree<String> dTree = new BinaryTree<String>("Is it in Europe", hTree,
iTree);

        BinaryTree<String> cTree = new BinaryTree<String>("Are you thinking of a famous F1
driver", fTree, gTree);
        BinaryTree<String> bTree = new BinaryTree<String>("Is it in the northern
hemisphere", dTree, eTree);

        // Now the root
        tree.setTree("Are you thinking of a famous monument or building", bTree, cTree);
    } // end createTree1

    public void interactiveQuestions(BinaryTree<String> tree, BinaryNodeInterface<String>
currentNode) {
        // Continues loop until system.exit condition or correct leaf is chosen by user
        System.out.println("Root node: " + tree.getRootNode() + " root data: " +
tree.getRootData());
        currentNode = tree.getRootNode(); // Set current node to root node

        try (Scanner scanner = new Scanner(System.in)) {
            String strInput, prompt;

            while (true) {
                while (!currentNode.isLeaf()) { // While current node is not a leaf - aka
quesiton
                    prompt = currentNode.getData() + "? (Y/N)"; // Ask the question
                    strInput = userSession.getUserInput(prompt);
                    strInput = strInput.toUpperCase();

                    switch (strInput) { // Update current node based on answer
                        case "Y":
                            currentNode = currentNode.getLeftChild();
                            break;
                        case "N":
                            currentNode = currentNode.getRightChild();
                            break;
                        default:
                            JOptionPane.showMessageDialog(null, "Input is invalid, enter
either Y or N");
                            break;
                    }
                }
                leafGuess(tree, currentNode); // We have reached a leaf, make the guess
            }
        }
```

```java
    }

    public static void leafGuess(BinaryTree<String> tree, BinaryNodeInterface<String>
currentNode) {
        System.out.println("Please use pop-up window to answer the question\n");
        String question = currentNode.getData() + "? (Y/N)"; // Make guess

        String strInput = userSession.getUserInput(question);
        strInput = strInput.toUpperCase();

        switch (strInput) { // Decide if we won, user makes new question & node or invalid
input
            case "Y":
                System.out.println("I win!, bringing you to the menu choices\n");
                JOptionPane.showMessageDialog(null, "I win!");
                userSession.menuChoice(tree, currentNode);
                break;
            case "N":
                System.out.println("I lose, please follow the instructions in the pop-up
window\n");
                newUserGenNode(tree, currentNode);
                break;
            default:
                System.out.println("Input is invalid, enter either Y or N");
                leafGuess(tree, currentNode);
                break;
        }
    }

    // Menu method for user
    public void menuChoice(BinaryTree<String> tree, BinaryNodeInterface<String>
currentNode) {
        String question = "Please select an option from the menu below \n1. Play (again)
\n2. Store the tree \n3. Load a stored tree \n4. Quit";
        String strInput = userSession.getUserInput(question);

        switch (strInput) {
            case "1": // Play again
                userSession.interactiveQuestions(tree, currentNode);
                break;
            case "2": // Store the tree
                treeIO.storeTree(tree);
                currentNode = tree.getRootNode();
                JOptionPane.showMessageDialog(null, "Tree stored");
                menuChoice(tree, currentNode);
                break;
            case "3": // Load a stored tree
                treeIO.loadTree();
                currentNode = tree.getRootNode();
                JOptionPane.showMessageDialog(null, "Tree loaded");
                menuChoice(tree, currentNode);
                break;
            case "4": // Quit
                System.exit(0);
```

```
                break;
            default:
                JOptionPane.showMessageDialog(null, "Invalid input, please enter a number
between 1 and 4");
                menuChoice(tree, currentNode);
                break;
        }

    }

    // if we hit a wrong guess we make new node and question
    public static void newUserGenNode(BinaryTree<String> tree, BinaryNodeInterface<String>
currentNode) {
        if (currentNode == null) {
            return;
        }
        String currentNodeAns = currentNode.getData(); // Get the current node data

        String prompt = "What is it that came to mind?";
        String nodeAnswer = userSession.getUserInput(prompt);
        nodeAnswer = "Is it " + nodeAnswer;

        prompt = "Type a question to differentiate between " + nodeAnswer + " and " +
currentNodeAns + "?";
        String nodeQuesiton = userSession.getUserInput(prompt);
        currentNode.setData(nodeQuesiton); // Put the question node in place of the
current node

        // Checking whether the answer is yes or no to the new question
        prompt = "Is the answer to the new question yes or no for " + nodeAnswer + " ?
(Y/N)";
        String inputStr = userSession.getUserInput(prompt);
        inputStr = inputStr.toUpperCase();

        // Setting the left and right child nodes
        // Current node data being moved down to either left or right child
        switch (inputStr) {
            case "Y":
                System.out.println("The left node: " + nodeAnswer + " right node: " +
currentNodeAns);
                currentNode.setLeftChild(new BinaryNode<String>(nodeAnswer));
                currentNode.setRightChild(new BinaryNode<String>(currentNodeAns)); //
Moving current node to the new
                                                                                    //
question node
                break;
            case "N":
                System.out.println("The left node: " + currentNodeAns + " right node: " +
nodeAnswer);
                currentNode.setLeftChild(new BinaryNode<String>(currentNodeAns));
                currentNode.setRightChild(new BinaryNode<String>(nodeAnswer));
                break;
            default:
                System.out.println("Input is invalid, enter either Y or N");
```

```
                break;
        }
        currentNode = tree.getRootNode(); // Resetting the current node to the root node

        userSession.menuChoice(tree, currentNode); // Returning to the menu choice
    }

    public static void displayTree(BinaryTree<String> tree) { // Displaying the tree
        int count = 1; // Counting the nodes
        Queue<BinaryNodeInterface<String>> queue = new LinkedList<>(); // Using a queue to
display the tree
        queue.add(tree.getRootNode());

        while (!queue.isEmpty()) {  // While the queue is not empty
            BinaryNodeInterface<String> node = queue.remove();
            System.out.println("Node " + count + ": " + node.getData()); // Displaying the
node
            count++;

            BinaryNodeInterface<String> left = node.getLeftChild();
            BinaryNodeInterface<String> right = node.getRightChild();

            if (left != null) {
                queue.add(left);
            }
            if (right != null) {
                queue.add(right);
            }
        }
        for (queue.size(); queue.size() > 0; queue.size()) {
            System.out.println(queue.remove()); // Displaying the tree
        }
    }

    public String getUserInput(String question) { //input validation
        String inputStr = JOptionPane.showInputDialog(null, question);

        if (inputStr == null) {
            System.exit(0);
        } else if (inputStr.equals("")) {
            JOptionPane.showMessageDialog(null, "Enter a valid input");
            return getUserInput(question);
        } else {
            return inputStr;
        }
        return inputStr;
    }

    public BinaryTree<String> loadTree() {
        BinaryTree<String> tree = new BinaryTree<String>();
        return tree;
    }

    public static void displayStats(BinaryTree<String> tree) {
```

```java
        if (tree.isEmpty())
            System.out.println("The tree is empty");
        else
            System.out.println("The tree is not empty");

        System.out.println("Root of tree is " + tree.getRootData());
        System.out.println("Height of tree is " + tree.getHeight());
        System.out.println("No. of nodes in tree is " + tree.getNumberOfNodes());
    } // end displayStats

}
```

treeIO.java

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class treeIO {

    static String filePath = System.getProperty("user.dir") + "\\";

    public void storeTree(BinaryTree<String> tree) {
        System.out.println("File path: " + filePath);

        // Serialization
        try {
            System.out.println("Saving tree");
            FileOutputStream file = new FileOutputStream("20QuestionTree.txt");
            ObjectOutputStream out = new ObjectOutputStream(file);

            out.writeObject(tree);

            out.close();
            file.close();
        } catch (IOException ex) {
            System.out.println("IOException caught");
            System.exit(1);
        }
    }

    public BinaryTree<String> loadTree() {
        BinaryTree<String> tree = null;

        // Deserialization
        try {
            System.out.println("Loading tree from file");
            FileInputStream file = new FileInputStream("20QuestionTree.txt");
            ObjectInputStream in = new ObjectInputStream(file);

            tree = (BinaryTree<String>) in.readObject();
            System.out.println("Tree: ");
            tree.inorderTraverse();

            in.close();
            file.close();

            return tree;
        } catch (IOException ex) {
            System.out.println("IOException caught");
            System.exit(1);
            return null;
        } catch (ClassNotFoundException ex) {
```

```
            System.out.println("ClassNotFoundException caught");
            System.exit(1);
            return null;
        }
    }
}
```