Paul Heather 22363463 – binary tree OOP

Problem Statement:

This is a problem that requires the implementation of binary nodes to create a binary tree. The user will be shown the root node first that will aks them a question to help guess their answer. If the user answers yes the program will move down and display th root nodes left child. If the user enters no it will display the right child. The program continues this logic for all nodes, however if the current node the user is on is a leaf node it is the games final guess and asks if it is the right answer. If it is the right answer the user can replay the game, save the tree, load a tree, or just quit the game. If the game guesses the users answer incorrectly, the game asks them for the proper answer and the defining question. It replaces the node the user is on with the question and puts the original data of the node on the new right child node as the wrong answer to the new question and the users inputted correct answer as the new left child node.

Analysis and Design:

Using the canvas code for the binarynode and binarynodeinterface classes. In the main class I will create a base binary tree that I will hard code to have a few nodes and leaf nodes for the user to play at first. I will do this using the setLeftChild() and setRightChild() and getLeftChild and getRightChild() methods. The traverseTree() method will be used to check the nodes of the tree corresponding to the users’ input. It will start at the root node. If the user enters yes it displays the left child node, if they enter no it dsiplays the right child node. This is as long as the current node is not a leaf node. If it Is a leaf node it displays the node data. This is the games final guess at the users’ answer. If they enter yes they are displayed their options of play again, save, load and quit. The corresponding method will be called based on their input. If the game guesses wrong they will be asked to enter a new question and answer which will be saved to the current node and the child nodes will be set as explained in the problem statement.

The save tree and load tree methods will use serialisation and deserialization respectfully. The save tree takes in a tree to be saved and the file to save it to. It saves the binary values to the file. The load method takes in a filename. It deserializes the values in the file taken in and saves ot to a BinaryNode<String>. That is then returned.

Code:

(used the canvas provided BinaryNode and BinaryNodeInterface code)

import java.util.Scanner;  
import java.io.\*;  
  
public class Main {  
 // initilaises the starting base tree  
 public static BinaryNode initialiseTree(){  
 // root node ( top of tree)  
 BinaryNode<String> rootNode = new BinaryNode<>("Is it an animal?");  
 BinaryNode<String> left1 = new BinaryNode<>("Is it a mammal?");  
 BinaryNode<String> right1 = new BinaryNode<>("Is it human?");  
  
 // sets left1 and right1 as the 2 child nodes of the rootNode  
 rootNode.setLeftChild(left1);  
 rootNode.setRightChild(right1);  
  
 // more nodes  
 left1.setLeftChild(new BinaryNode<>("Does it have 4 legs?"));  
 left1.setRightChild(new BinaryNode<>("Is it a bird?"));  
 right1.setLeftChild(new BinaryNode<>("Are they male?"));  
 right1.setRightChild(new BinaryNode<>("Is it an alien?"));  
  
  
 left1.getLeftChild().setLeftChild(new BinaryNode<>("Is it a dog?"));  
 left1.getLeftChild().setRightChild(new BinaryNode<>("Is it a kangaroo?"));  
  
 right1.getLeftChild().setLeftChild(new BinaryNode<>("Is it Jake Gyllenhaal?"));  
 right1.getLeftChild().setRightChild(new BinaryNode<>("Is it Michelle Obama?"));  
  
 return rootNode;  
  
  
 }  
  
 // method to traverse through the nodes of the tree  
 public static void traverseTree(BinaryNode<String> tree) {  
 // keeps track of the users current node  
 BinaryNodeInterface<String> currentNode = tree;  
 // scans in users input  
 Scanner userScanner = new Scanner(System.*in*);  
  
 // while the user is not on a leaf node it checks if their input is yes or no and goes left for yes and right for no otherwise prints invalid input and displays the node again  
 while (! currentNode.isLeaf()) {  
 System.*out*.println(currentNode.getData());  
 String userInput = userScanner.nextLine();  
 if (userInput.equals("yes") || userInput.equals("Yes") || userInput.equals("YES")) {  
 if (currentNode.hasLeftChild()) {  
 currentNode = currentNode.getLeftChild();  
 }  
  
 }  
 else if (userInput.equals("no") || userInput.equals("No") || userInput.equals("NO")) {  
 if (currentNode.hasRightChild()) {  
 currentNode = currentNode.getRightChild();  
 }  
  
 }  
 else {  
 System.*out*.println("Invalid input");  
 }  
  
 }  
 // if the user is on a leaf node and they enter yes the game has guessed correct and gives them the 4 options  
 System.*out*.println(currentNode.getData());  
 String userInput = userScanner.nextLine();  
 if (userInput.equals("yes") || userInput.equals("Yes") || userInput.equals("YES")) {  
 System.*out*.println("I guessed correct !!");  
 // boolean to make sure they put in either 1,2,3 or 4  
 boolean validAnswer = false;  
 while(!validAnswer) {  
 System.*out*.println("Would you like to:\n1. Play again\n2. Store this tree\n3. Load a stored tree\n4. Quit");  
 String userInputCorrect = userScanner.nextLine();  
 if (userInputCorrect.equals("1")) {  
 // calls the traverse tree function again  
 *traverseTree*(tree);  
 validAnswer = true;  
 } else if (userInputCorrect.equals("2")) {  
 System.*out*.println("Storing.....");  
 validAnswer = true;  
 // calls save tree mehtod  
 *saveTree*(tree, "tree.ser");  
 } else if (userInputCorrect.equals("3")) {  
 System.*out*.println("Loading.....");  
 // loads in saved tree and traverses it  
 BinaryNode<String> loadedTree = *loadTree*("tree.ser");  
 *traverseTree*(loadedTree);  
 validAnswer = true;  
 } else if (userInputCorrect.equals("4")) {  
 // ends loop  
 System.*out*.println("Closing");  
 validAnswer = true;  
 } else {  
 System.*out*.println("Enter a valid number");  
 validAnswer = false;  
 }  
 }  
 }  
  
 // if the game guesses incorrectly user enters no :  
 else if (userInput.equals("no") || userInput.equals("No") || userInput.equals("NO")) {  
 System.*out*.println("Oops I guessed wrong !!");  
 System.*out*.println("What was the correct answer");  
 //scans what the right answer should have been  
 String userInputWrongL = userScanner.nextLine();  
 System.*out*.println("Enter the distinguishing question to help guess next time: ");  
 // scans the new question  
 String userInputWrongQ = userScanner.nextLine();  
 // saves the data of the current node  
 String userInputWrongR = currentNode.getData();  
  
 // puts the correct answer into node form  
 BinaryNode<String> userInputWrongLNode = new BinaryNode<>("Is it a "+userInputWrongL+"?");  
 // puts the string taken from the current node back into node form  
 BinaryNode<String> userInputWrongRNode = new BinaryNode<>(userInputWrongR);  
  
 // changes the current node to the question, right node to the old nodes data and the left child node to the users correct answer  
 currentNode.setData(userInputWrongQ);  
 currentNode.setLeftChild(userInputWrongLNode);  
 currentNode.setRightChild(userInputWrongRNode);  
  
 // calls the save tree function automatically and lets the user traverse the tree again  
 *saveTree*(tree, "tree.ser");  
  
 *traverseTree*(tree);  
 }  
  
 }  
  
 // method to save a whole tree  
 // method takes in a tree and file name  
 public static void saveTree(BinaryNode<String> tree, String filename) {  
 try {  
 //creates a new fileoutputstream  
 FileOutputStream fileOut = new FileOutputStream(filename);  
 // used to take in serialized version of filename  
 ObjectOutputStream out = new ObjectOutputStream(fileOut);  
 // writes serialized tree to out  
 out.writeObject(tree);  
 out.close();  
 fileOut.close();  
 System.*out*.println("Tree has been saved to " + filename);  
 //exception handling  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 // Method to load the tree using deserialization  
 public static BinaryNode<String> loadTree(String filename) {  
 BinaryNode<String> loadedTree = null;  
 try {  
 // stores the binary tree  
 FileInputStream fileIn = new FileInputStream(filename);  
 // takes in deserialized version  
 ObjectInputStream in = new ObjectInputStream(fileIn);  
 //puts deserialized tree into loadedTree tree  
 loadedTree = (BinaryNode<String>) in.readObject();  
 in.close();  
 fileIn.close();  
 System.*out*.println("Tree has been loaded from " + filename);  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return loadedTree;  
 }  
  
  
  
  
  
 public static void main(String[] args) {  
  
 BinaryNode<String> baseTree = *initialiseTree*();  
 *traverseTree*(baseTree);  
 }  
  
}

Testing:

When the user runs the application the base tree is displayed and the user traverses it. When they get to the end the 4 options are displayed:

A screenshot of a computer program

Description automatically generated

When the user enters 1 it runs the same tree again:

If the games guess is wrong the user puts in their answer and question:

A screenshot of a computer program

Description automatically generated

When the user enters the new question and answer the tree saves automatically and the user can play again:

A screenshot of a computer program

Description automatically generated

Storing the tree :

A screenshot of a computer program

Description automatically generated

When the user chooses the option of loading the tree:

A screenshot of a computer

Description automatically generated

When the user chooses 4 close the game;

A screenshot of a computer program

Description automatically generated