Assignment 5 – Expandable Binary Tree Guessing Game

Problem Statement:

The example code that is provided will be used in this assignment and will make up the majority of the code. In order to complete this assignment to the best of quality I must understand each and every example code provided in order to know how I can create the project effectively, before I start I will run through and interpret everything. I must research the best way to store the data, I must have a strong knowledge on trees, binary trees and nodes along with being able to identify and define parts of the tree structures. My job is to create methods for serialising and deserializing objects so that I can save trees and load saved trees in the hard drive. I will create a method for printing a tree to the screen for testing in the form of JOptionPanes. I will create a method for obtaining user input when traversing through the levels while creating a method for traversing the nodes so that I can ask the user a new question based on their previous response. While making these I must take user errors into account and handle them appropriately. I must then test the code extensively to ensure everything works effectively.

Analysis and Design Notes:

The example coding files that I used for this assignment are:
BinaryNode.java
BinaryNodeInterface
BinaryTree
BinaryTreeInterface
BinaryTreeDemo
TreeInterface

I used the BinaryTreeDemo to create my tree using code and I made a file class so that I can work out how to serialize and deserialize the data.

GuessingGame.java

This is where we will execute our code. In this file we will create our initial tree structure and create our question method and get Input method in order to make the code have sufficient functionality. Our question method builds on the main method template provided in the assignment sheet.

Method: createTree1 builds the initial tree structure with 4 levels and 6 leaves. I use String values to create the branches. I can populate the tree by declaring new instances of the BinaryTree class and linking it to a node in the previous level or if I'm starting the tree it will create the root node.

Method: Question is our main method that will run through the process while it isn't broken, while the current node isn't a leaf we will ask the question and either continue down the left or right of the data structure. We must also take the user errors into account and handle them appropriately. Once we finish the sequence we handle the final steps outside the loop. If the answer is correct, we ask the user questions for them to proceed in the manner they want but if it's wrong we can expand the tree by asking the user to add a relevant question for their answer.

Method: getInput: is used in both GuessingGame.java and FilePanel.java. The purpose of this method is the provoke the JOPTIONPANE so that we can get a user input. If the user presses cancel we exit the loop or if the user doesn't enter a valid input we prompt them to enter one with a message but if the user enters the correct input the method return the input.

FilePanel.java

This will handle the file input and output. The get input uses the JOPTIONSPANE to prompt the user to enter the string value. This method will cancel the program if the user presses the cancel button or prompt the user to enter a valid input if it's invalid. This class is also makes the store and load tree methods. This is used through file serialisation and deserialization

Method: getInput gets the input from the user and displays it using the JOptionPane. It will also handle input errors. Same as the getInput from the GuessingGame.java although the getInput method in this code is handling the user input for after the user has gone through the tree successfully and we reach the prompt where the user can manually choose to load/store/exit/play again. Both classes use this method but for different parts of the code.

Method: storeTree this method focuses on serializing the file, it coverts the binary tree data object into a data structure and prompts the user to enter a file name, this file will then be saved into the selected repository to be loaded at any time.

Method: loadTree will load the data structure with the updated tree structure from the selected file. It will de-serialise the file and return the binary tree.

Code :

BinaryTreeDemo:

```
import javax.swing.*;

public class GuessingGame
{
   public static void main(String[] args)
   {
      // Create a tree
      System.out.println("Constructing a test tree ...");
      BinaryTree<String> testTree = new BinaryTree<String>();
      createTree1(testTree);

      // 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 test tree, printing
each node when visiting it ...");
      testTree.inorderTraverse();

      question(testTree);

   } // end of File

   public static void createTree1(BinaryTree<String> tree)
   {
      BinaryTree<String> hTree = new BinaryTree<>("Is it a dog?");
      BinaryTree<String> iTree = new BinaryTree<>("Is it a lion?");
      BinaryTree<String> jTree = new BinaryTree<>("Is it an eagle?");
      BinaryTree<String> kTree = new BinaryTree<>("Is it a snake?");
      BinaryTree<String> lTree = new BinaryTree<>("Is it a car?");
```

```
      BinaryTree<String> mTree = new BinaryTree<>("Is it a house?");
      BinaryTree<String> nTree = new BinaryTree<>("Is it a laptop?");
      BinaryTree<String> oTree = new BinaryTree<>("Is it a hammer?");

       // First the leaves
      BinaryTree<String> dTree = new BinaryTree<String>("Is it a pet?",
hTree, iTree);
      BinaryTree<String> eTree = new BinaryTree<String>("Is it a bird?",
jTree, kTree);
      BinaryTree<String> fTree = new BinaryTree<String>("Does it have
wheels?", lTree, mTree);
      BinaryTree<String> gTree = new BinaryTree<String>("Is it
technology?", nTree, oTree);

      // Now the subtrees joining leaves:
      BinaryTree<String> bTree = new BinaryTree<String>("Is it a mammal?",
dTree, eTree);
      BinaryTree<String> cTree = new BinaryTree<String>("Is it an object?",
fTree, gTree);

      // Now the root
      tree.setTree("Are you thinking of an animal?", bTree, cTree);
   }

   public static void question(BinaryTree<String> tree){
      // loop until process is broken or finished
      while(true){
         BinaryNodeInterface<String> currentNode = tree.getRootNode();
         String answer;
         while(!currentNode.isLeaf()){
            answer = FilePanel.getInput(currentNode.getData());
            if(answer.equals("yes")){
               currentNode = currentNode.getLeftChild();
            } else if (answer.equals("no")) {
               currentNode = currentNode.getRightChild();
            } else {
               JOptionPane.showMessageDialog(null, "Please enter yes or no.
\n");
            }
         }
         answer = FilePanel.getInput(currentNode.getData());

         if(answer.equals("yes")){
            JOptionPane.showMessageDialog(null, "The tree guessed
correctly!\n");
            answer = FilePanel.getInput("Would you like to:\n\t1. Play
again.\n\t2. Store the tree.\n\t3. Load a stored tree\n\t4. Quit.\n");
            if (answer.equals("1")) currentNode = tree.getRootNode();
            if (answer.equals("2")) {
               FilePanel.storeTree(tree);
               currentNode = tree.getRootNode();
            }
            if(answer.equals("4"))
               System.exit(0);
         }
         else if (answer.equals("no")){
            answer = FilePanel.getInput("I don't know: what is the correct
answer?\n");
            currentNode.setLeftChild(new BinaryNode<>("Is it a " +
answer));
            currentNode.setRightChild(new
```

```
BinaryNode<>(currentNode.getData()));
            answer = FilePanel.getInput("Distinguishing question?\n");
            currentNode.setData(answer);
            currentNode = tree.getRootNode();
         } else {
            JOptionPane.showMessageDialog(null, "Please enter yes or
no.\n");
         }
      }
   }
   // handling user errors in the user input
   public static String getInput(String question) {
      String input = JOptionPane.showInputDialog(null, question);
      if (input == null)
               System.exit(0);
      else if (input.equals("")){
         JOptionPane.showMessageDialog(null, "Please enter a valid input");
         return getInput(question);
      } else {
         return input;
      }
      return input;
   }
   public static void displayStats(BinaryTree<String> tree)
   {
      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
}
```

FilePanel.java:

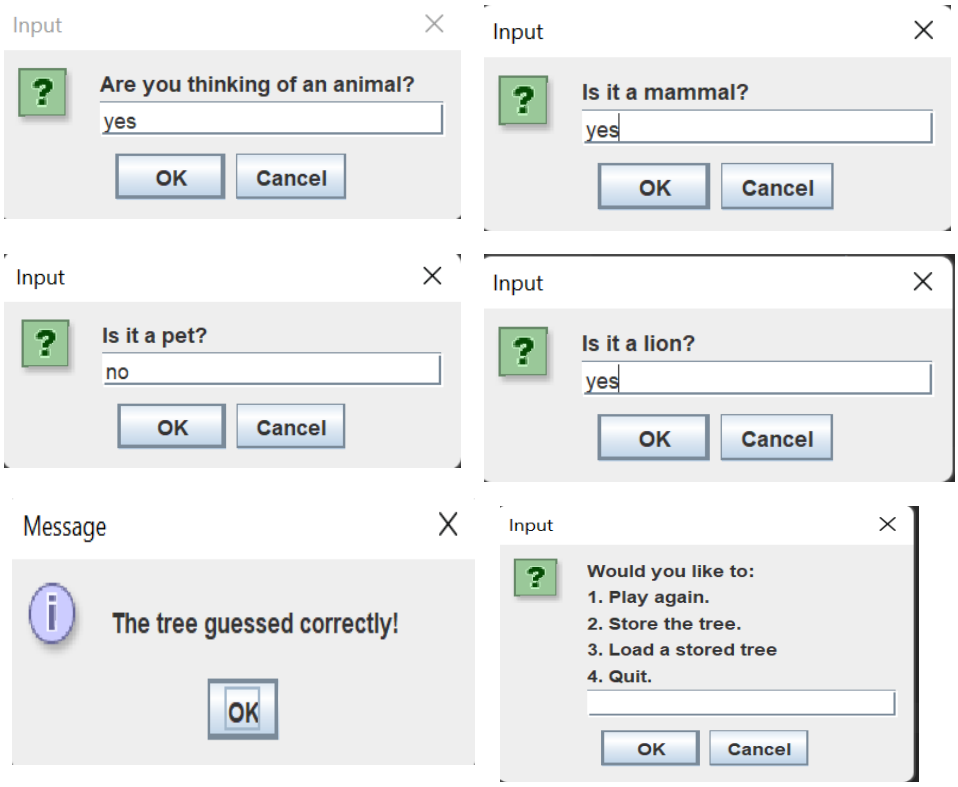
```
import java.io.*;
import java.util.Scanner;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class FilePanel extends JPanel {

    // handles the user's input
    public static String getInput(String question) {
        String input = JOptionPane.showInputDialog(null, question);
        if (input == null) {
            System.exit(0);
        } else if (input.equals("")) {
            JOptionPane.showMessageDialog(null, "Please enter a valid
input");
            return getInput(question);
        } else {
            return input;
        }
        return input;
```

```
    }

    // saves and stores the completed tree inside a file by serialization
    public static void storeTree(BinaryTree<String> tree) {
        String question = "Enter the name of the file to store in the
directory: " + System.getProperty("user.dir");

        try {
            FileOutputStream fileOutput = new
FileOutputStream(getInput(question));
            ObjectOutputStream objectOutput = new
ObjectOutputStream(fileOutput);
            objectOutput.writeObject(tree);
            objectOutput.close();
            fileOutput.close();
        }
        catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Please enter a valid
filename");
            System.out.println("IOException caught: " + e.getMessage());
            storeTree(tree);
        }
    }
    public static BinaryTree<String> loadTree() {
        BinaryTree<String> tree = null;
        String question = "Enter the name of the file to load from the
directory: " + System.getProperty("user.dir");

        try {
            FileInputStream fileInput = new
FileInputStream(getInput(question));
            ObjectInputStream objectInput = new
ObjectInputStream(fileInput);
            tree = (BinaryTree<String>) objectInput.readObject();
            objectInput.close();
            fileInput.close();
            return tree;
        }
        catch (IOException | ClassNotFoundException e) {
            JOptionPane.showMessageDialog(null, "Please enter a valid
filename");
            System.out.println("Exception caught: " + e.getMessage());
            return loadTree();
        }
    }
}
```

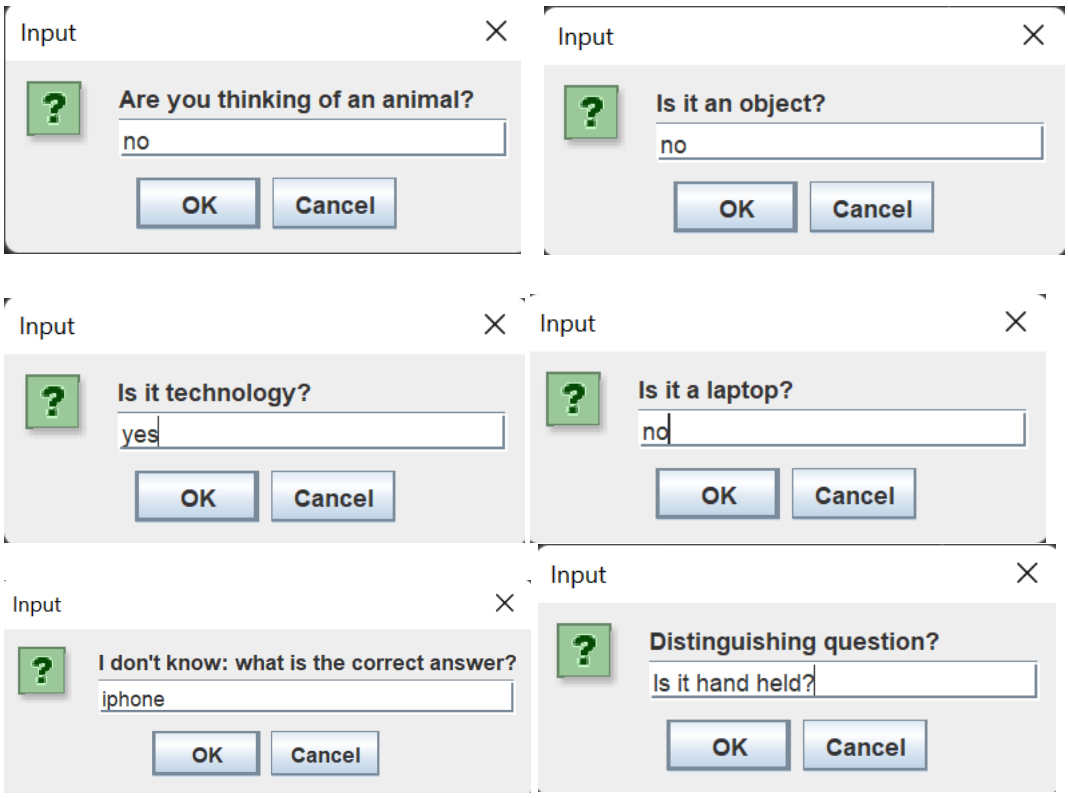
Testing: I tested every aspect of the guessing game

```
Some statistics about the test tree ...
The tree is not empty
Root of tree is Are you thinking of an animal
Height of tree is 4
No. of nodes in tree is 15
```

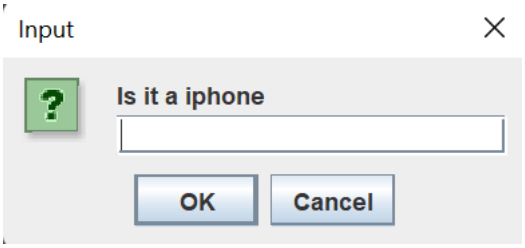
Test1: successful guessing game

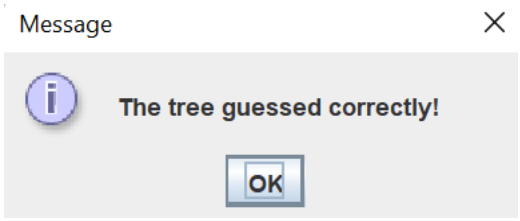


Test2: unsuccessful guess and storing the new tree in a file that we can later load

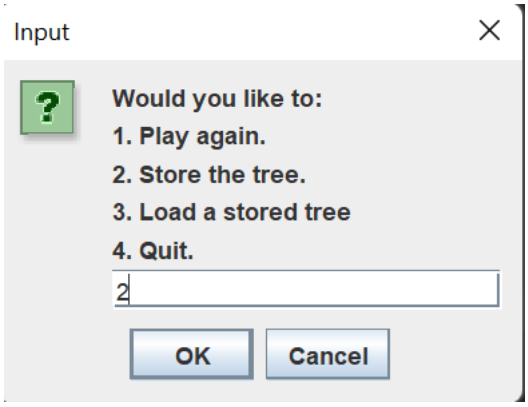


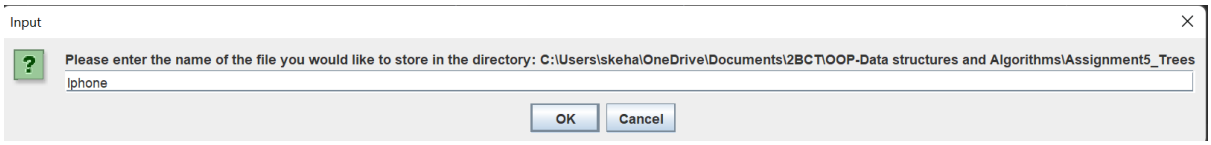
Repeat with new node: as you see we successfully added the iPhone node, this file will be saved in the folder to be used at a later time.

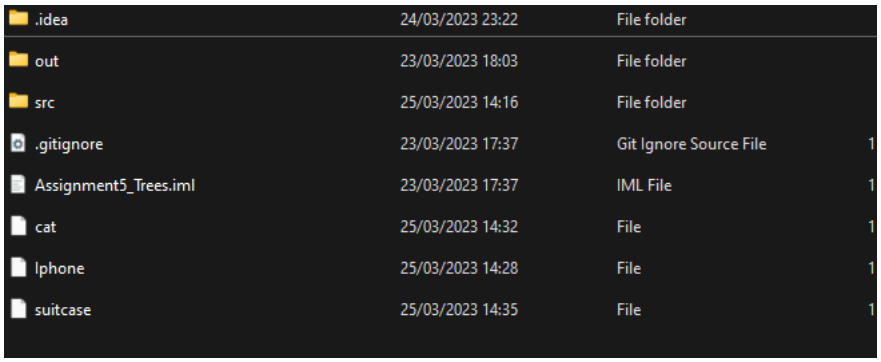




Storing the tree after a successful run:







As you see we have successfully saved the new files into our folder through serialization.

Errors that the user inputs:

1. Blank input field:

2. Incorrect spelling: