Cathal McSweeney : 19731485

OOP ASSIGNMENT FIVE – EXPANDABLE BINARY TREE

Problem Statement:

For this assignment we are faced with creating a tree of objects that tries to guess what a user is thinking of. It is to be expandable so that if the tree is unable to guess what a user is thinking of then through a series of prompts a user is able to add to the tree. The tree is expandable in that there is no limit to the number of additions that a user can make to the tree. Once a correct guess is found the user is then prompted to make a selection choosing to, play again, save the tree, load a tree, or exit. Incorporating the various Binary Tree classes as provided from blackboard, we altered the "BinaryTreeDemo" class in order to achieve this.

Analysis and Design Notes

Pseudocode :

- Main Method
  - Initialize binary tree object
  - Create starting binary tree by calling 'createTree' method
  - Prompt user to load saved tree or not
  - Display stats on tree being used
  - Display nodes in tree
  - Call questions method to start yes no questions for user

- Method One
  - Initialize 8 leaf nodes
  - Initialize 4 leaf node parents
  - Initialize 2 parents of previous 4 nodes
  - Initialize root node

- Method Two
  - Given "DisplayStats" Method

- Method Three
  - Load root node of the tree

```
While(true){
        String answer
        While(not a leaf){
                answer = Get user input(current node)
                if(answer = yes)
                        get left node
                else if(answer = no)
                        get right node
                else
                prompt user again to enter yes/no
```

```
        }

        Answer = Get user input(current node)

        If(yes){
                Print out correct guess from tree
                Answer = Get user input (Prompt user to make 1 of 4 options)

                If(1){
                        Start again
                }

                If(2){
                        Save tree
                        Start tree from beginning
                }

                If(3){
                        Load tree
                        Start tree from beginning
                }

                If(4){
                        exit
                }

        }

        Else if(no){
                Answer = Get user input (prompt user for correct answer)
                Set new left node
                Set new right node(current node)
                Answer = Get user input (distinguishing question)
                Set current node(Answer)
                Start from start of tree
        }
        Else{
                Prompt user to enter yes/no
        }
}
```

- Method Four

```
Public static String userInput(String question)
    String input = prompt(question)
    If(input = “”){
            Prompt user to enter valid input
            Call userInput method again
    }
    Else{
            Return input
    }
```

Flow of control

The program will need to initialize a binary tree that is empty in the main method. It then needs to call on the create tree prompt the user to identify whether they wish to load a previously saved tree or not. Calls on a method that displays the statistics of the tree being used. This is to ensure that the correct tree is being loaded if the user selects it to be. For testing purposes the main method also prints out all the nodes in the order they are visited to ensure all nodes are present in the tree, particularly if a tree is loaded. The main method will finally then call the method that questions the user.

In the questions method a current node is initialized with the root node of the tree. An infinite loop is then created which contains a nested loop that iterates through all the nodes of the tree until it reaches a leaf. During this the user will be presented a question in the form of a JOption Dialog box displaying a question for the user to reply with a yes or no to. Depending on the answer of the user the current node will be altered to contain either its left child or its right child.

When a current node is a leaf the answer that the user gives will be very different. If a yes answer is provided a user is told that the tree has correctly guessed what they were thinking. Then prompts the user to make 1 of 4 options. (1)start again (2)save tree (3)load tree (4)exit. If a user selected no, they are told that the tree doesn’t know what the answer is and asks the user to input the correct answer. Which it then sets the left child of the current node to this new answer, sets the right child of the currentNode to contain the data of the current node. The user is then prompted to enter a distinguishing question which is then used to set the data of the currentNode to. The program then start again.

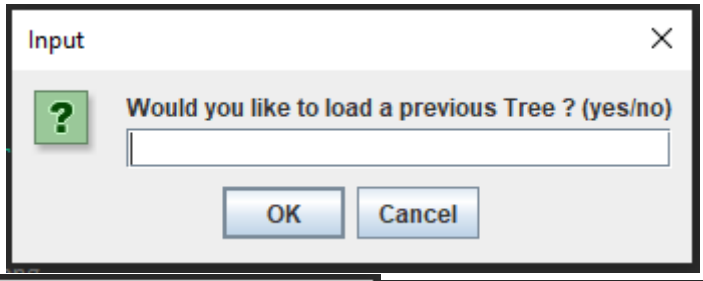
Testing:

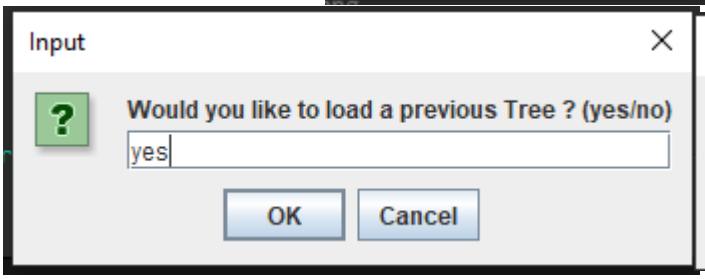
In testing the code the first thing we would look at is the output to the console to ensure that the tree being used is the correct tree. This is achieved through the displayStats method and the inorderTraverse method. As we can see below the tree is not empty, the root node is displayed along with the height of the tree and the number of nodes within it. Followed by the contents of every node in the order they are visited.

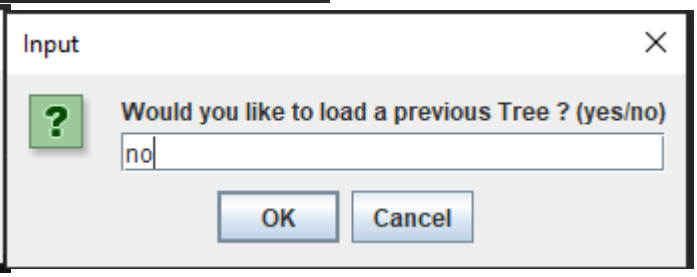
```
Constructing a test tree ...

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

In-order traversal of the test tree, printing each node when visiting it ...
Is it a dog?
Does it have legs?
Is it a Snake?
Is it a land animal?
Is it a salmon?
Does it live in the water?
Is it a crow?
Are you thinking of an animal?
Is it a car?
Is it a vehicle?
Is it a drill?
Is it mechanical?
Is it a Playstation 5?
Is it a Computer?
Is it a Brick?
```

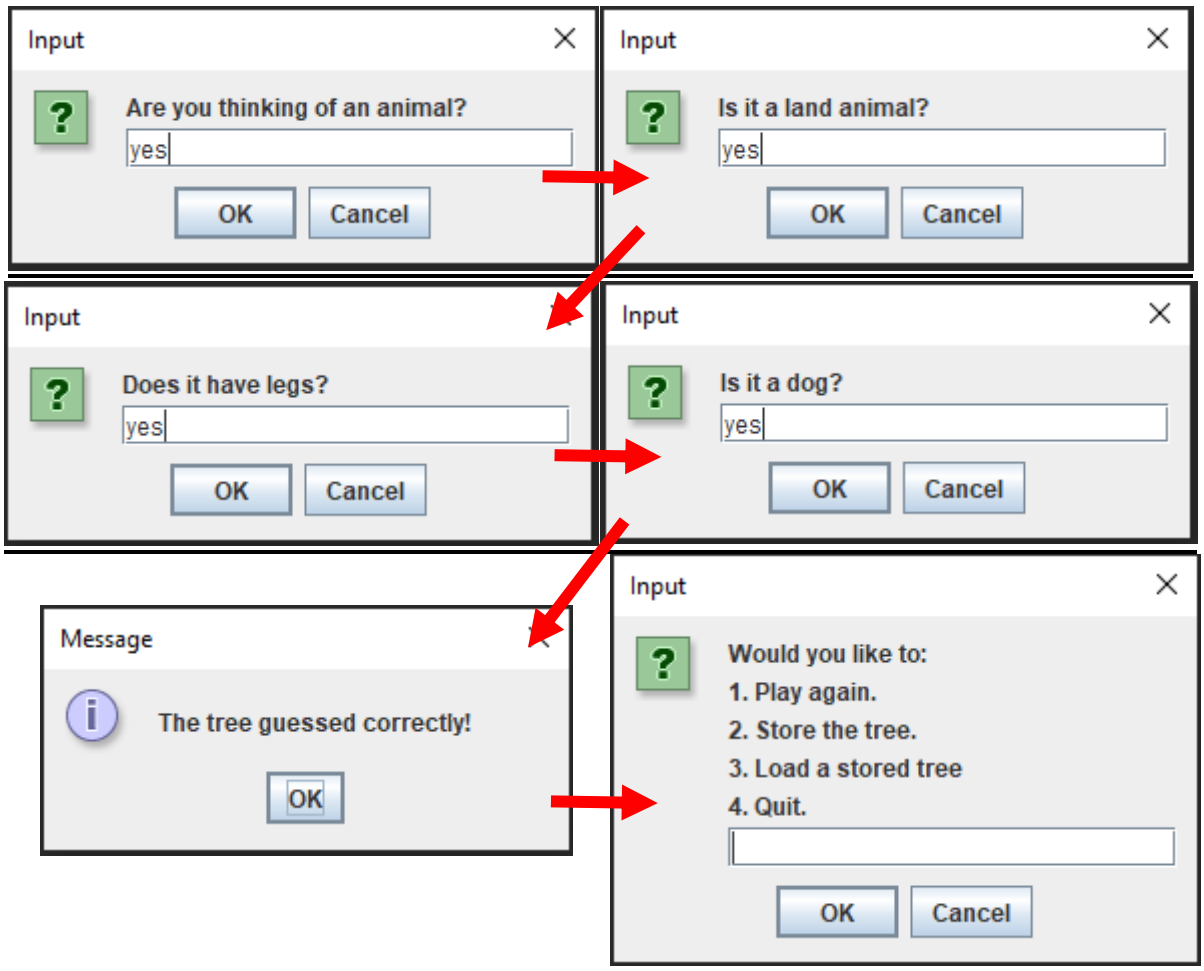


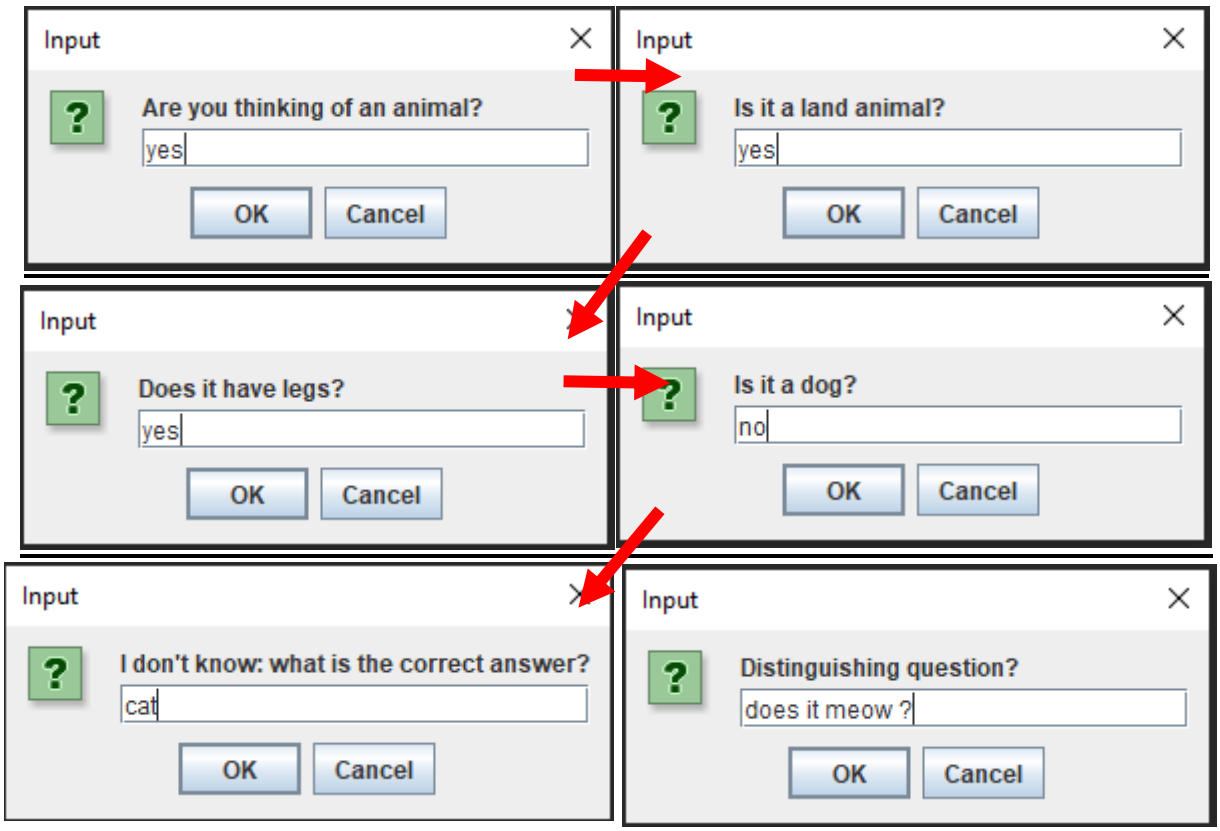




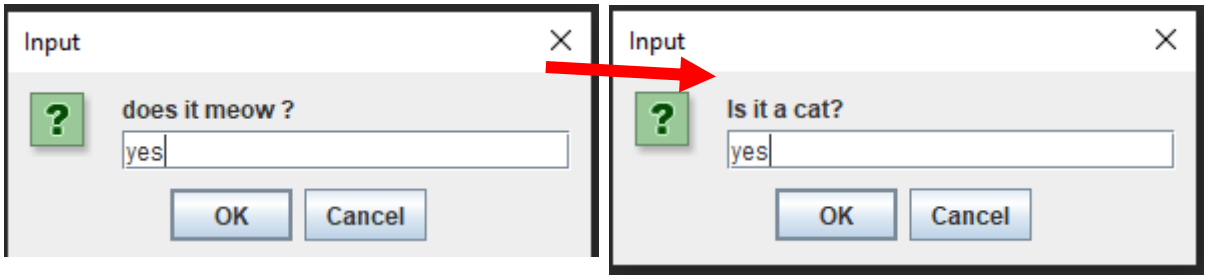

```
In-order traversal of the test tree, printing each node when visiting it ...
Is it a dog?
Does it have legs?
Is it a Snake?
Is it a land animal?
Is it a salmon?
Does it live in the water?
Is it a BIG TEST ANSWER FOR TESTING?
THIS IS TO BE SAVED & LOADED LATER
Is it a crow?
Are you thinking of an animal?
Is it a car?
Is it a vehicle?
Is it a drill?
Is it mechanical?
Is it a Playstation 5?
Is it a Computer?
Is it a Brick?
```

```
In-order traversal of the test tree, printing each node when visiting it ...
Is it a dog?
Does it have legs?
Is it a Snake?
Is it a land animal?
Is it a salmon?
Does it live in the water?
Is it a crow?
Are you thinking of an animal?
Is it a car?
Is it a vehicle?
Is it a drill?
Is it mechanical?
Is it a Playstation 5?
Is it a Computer?
Is it a Brick?
```


Input
Are you thinking of an animal?
yes
OK
Cancel
Input
Is it a land animal?
yes
OK
Cancel
Input
Does it have legs?
yes
OK
Cancel
Input
Is it a dog?
yes
OK
Cancel
Message
The tree guessed correctly!
OK
Input
Would you like to:
1. Play again.
2. Store the tree.
3. Load a stored tree
4. Quit.
OK
Cancel



After entering the new node and its children the user is prompted with the rootnode of the tree with these new nodes included as can be seen below



CODE

```
import java.io.IOException;

import javax.swing.JOptionPane;

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

```
		createTree1(testTree);

		String loadtree = JOptionPane.showInputDialog(null, "Would you like to load
a previous Tree ? (yes/no)\n");
		if(loadtree.toLowerCase().equals("yes")) {
			testTree = FileSaveLoad.loadTree();
		}

		// 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();

		questions(testTree);
	}

	public static void createTree1(BinaryTree<String> tree){
		// First the leaves
	BinaryTree<String> hTree = new BinaryTree<>("Is it a dog?");
	BinaryTree<String> iTree = new BinaryTree<>("Is it a Snake?");
	BinaryTree<String> jTree = new BinaryTree<>("Is it a salmon?");
	BinaryTree<String> kTree = new BinaryTree<>("Is it a crow?");
	BinaryTree<String> lTree = new BinaryTree<>("Is it a car?");
	BinaryTree<String> mTree = new BinaryTree<>("Is it a drill?");
	BinaryTree<String> nTree = new BinaryTree<>("Is it a Playstation 5?");
	BinaryTree<String> oTree = new BinaryTree<>("Is it a Brick?");
	// Now the subtrees joining leaves:
	BinaryTree<String> dTree = new BinaryTree<>("Does it have legs?", hTree, iTree);
	BinaryTree<String> eTree = new BinaryTree<>("Does it live in the water?", jTree,
kTree);
	BinaryTree<String> fTree = new BinaryTree<>("Is it a vehicle?", lTree, mTree);
	BinaryTree<String> gTree = new BinaryTree<>("Is it a Computer?", nTree, oTree);

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

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

	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

        public static void questions(BinaryTree<String> tree) throws IOException {
      // Looping over until loop is broken or an option is chosen and the loop is reset.
      BinaryNodeInterface<String> currentNode = tree.getRootNode();
      while (true){
        String ans;
        while(!currentNode.isLeaf()) {
          // Ask the question and update current node
          // based on the answer
          ans = userInput(currentNode.getData());
          // Comparing user input and leading them down different node children depending
on their answer
          if (ans.toLowerCase().equals("yes")) {
            currentNode = currentNode.getLeftChild();
          } else if (ans.toLowerCase().equals("no")){
            currentNode = currentNode.getRightChild();
          } else {
            JOptionPane.showMessageDialog(null, "Please enter yes or no.\n");
          }
        }

        // At the leaf: got an answer that is either right or wrong
        // Printing out the question
        ans = userInput(currentNode.getData());
        // Taking input and determining if yes or no, if yes correct answer found
        // requests user to select one of four options
        if (ans.equals("yes")) {
          JOptionPane.showMessageDialog(null, "The tree guessed correctly!\n");
          ans = userInput("Would you like to:\n\t1. Play again.\n\t2. Store the tree.\n\t3. Load
a stored tree\n\t4. Quit.\n");
          if (ans.equals("1")) {
                currentNode = tree.getRootNode();
          }
          if (ans.equals("2")) {
                FileSaveLoad.storeTree(tree);
```

```
                currentNode = tree.getRootNode();
            }
            if (ans.equals("3")) {
                tree = FileSaveLoad.loadTree();
                currentNode = tree.getRootNode();
            }
            if (ans.equals("4")) System.exit(0);
        }
        // Adding new node to the tree when no answer is found
        else if (ans.equals("no")){
            ans = userInput("I don't know: what is the correct answer?\n");
            currentNode.setLeftChild(new BinaryNode<>("Is it a " + ans+ "?"));
            currentNode.setRightChild(new BinaryNode<>(currentNode.getData()));
            ans = userInput("Distinguishing question?\n");
            currentNode.setData(ans);
            currentNode = tree.getRootNode();
        }
        else {
            JOptionPane.showMessageDialog(null, "Please enter yes or no.\n");
        }
    }
  }

        //gets user input for all JOption prompts
        public static String userInput(String question) {
    String in = JOptionPane.showInputDialog(null, question);
    if (in.equals("")){
        JOptionPane.showMessageDialog(null, "Please enter a valid input");
        return userInput(question);
    }
    else {
        return in;
    }
  }
}
```

FileSaveLoad Class

```
import javax.swing.*;
import java.io.*;

public class FileSaveLoad {

        public static void storeTree(BinaryTree<String> tree) {
    try{
        //Saving of object in a file
```

```
            FileOutputStream file = new FileOutputStream(System.getProperty("user.dir") +
"\\savedTree.txt");
            ObjectOutputStream out = new ObjectOutputStream(file);

            //wrtie the tree object to a txt file
            out.writeObject(tree);
            out.close();
            file.close();
        }

        catch(IOException ex){
            System.out.println("IOException is caught");
            storeTree(tree);
        }
    }

            public static BinaryTree<String> loadTree() throws IOException {
        BinaryTree<String> tree = null;//empty tree object

        try {
            // Reading the object from a txt file
            FileInputStream file = new FileInputStream(System.getProperty("user.dir") +
"\\savedTree.txt");
            ObjectInputStream in = new ObjectInputStream(file);

            //create the tree object from the txt file
            tree = (BinaryTree<String>) in.readObject();

            in.close();
            file.close();
            return tree;
        }

        catch(ClassNotFoundException ex){
            System.out.println("ClassNotFoundException is caught");
            return loadTree();
        }
    }
}
```