Assignment 5: Expandable Binary Tree Guessing Game

Problem Statement

This assignment can use the majority of the code examples used in the lectures. All I will need to do is develop my own strategies in serializing/deserializing objects so that I can save trees to the disk, develop a way to print a tree to the screen for testing, get input from the user for when they are editing a node and develop a way of traversing the nodes to I can ask the user a new question based on their previous answer.

Analysis and Design Notes

Method: Tree Printing
I imagine that I will be able to amend the existing inorderTraverse function to suit my needs to do this as the function already visits all nodes in order so as it visits the nodes, I will likely add them to an 2 dimensional array depending on the height of that node. E.g. if they have a height of 1 that means they are the leaf nodes, and I will add them to the top of the array, this will mean I will need to flip the array though.

Method: Array flip
Take in array from inordertraverse and flip it using a for loop, loop through array from inordertraverse backwards and add it to new array starting from top.

Method: Display tree
Loop through every row of array, then loop through every column in that row and print it to the screen.

Method: get Input
Take input in from user using JOptionPane, make sure to account for the cancel button and parse input to make sure its not null.

Method: Questioning user

```
While (true) {
        While (!currentNode.isLeaf()) {
                ans = getInput(currentNode.getData());
                If (ans == “yes”) go to left child
                Else if (ans == “no”) go to right child
                else getInput(currentNode.getData()); // As input wasnt valid
        }
        // This only runs when we've reached a leaf
        // Asking leaf question (Last question), children lead to yes which allows them to edit the
node or no which says the tree wins.
        Ans = getInput(currentNode.getData());
        If (ans == "yes")  { //Give all options such as saving tree, loading tree, exiting
                Ans = getInput("Options");
                If (ans == "1") currentNode = tree.getRootNode();
                If (ans == "2") Fileio.store();
                // etc..
        } else if (ans == "no) {
                //admit tree is wrong
                ans = getInput("new answer?");
                // set the right and left child accordingly
                leftChild = ans;
                rightChild = previous answer;
                ans = getInput("new question?");
                currentNode.setData(ans);
        } else {
                showMessage("enter valid input");
        }
}

// File IO
// Serialize & Store Tree
Try {
        FileOutputStream file = new FileOutputStream(getInput(question)); // Ask user what they
want to call the file
        ObjectOutputStream out = new ObjectOutputStream(file);

        out.writeObject(tree);

        Close files
} catch (exception) {
        showMessage("exception");
}

// Deserialize & load tree
Try {
        FileInputStream file = new FileInputStream(getInput(question)); // Ask user the file name
        ObjectInputStream in = new ObjectInputStream(file);

        Tree = in.readObject();
        Close files

        Return tree;
} catch (exception) {
        showMessage(“exception”);
}
```

There are a few other methods, such as the one to create the tree, but I will just use the one that Frank used in the lecture.

Code

BinaryTreeDemo.java

```
/* CT2109 NUI Galway:
 * Class to demonstrate the use of BinaryTree code.
 * Based on code by Carrano & Savitch.
 * @author Michael Madden. Improved Upon By Conor Mc Govern
 */
import javax.swing.*;

public class BinaryTreeDemo
{
        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 -> Printing tree
        // Testing the tree has been deserilized from file/set from createTree1 correctly
        System.out.println("\nPrinting tree representation ...");
        displayTree(testTree);

        // Asking questions
        question(testTree);

        } // end of main
```

```
public static String[][] flipArray(BinaryTree<String> tree, String[][] lines) {
String[][] tmp = new String[tree.getHeight()][tree.getHeight() * tree.getHeight()];
int x = tree.getHeight() - 1;
for (int i = 0; i <= tree.getHeight() - 1; i++) {
tmp[i] = lines[x];
x--;
}
return tmp;
}

public static void displayTree(BinaryTree<String> tree) {
// Calling inorderTraverse to get array of all nodes in order of level
// Reversing the array as the tree is given in reverse order.
String[][] lines = flipArray(tree, tree.inorderTraverse());

// Displaying each row of the array which is every line of nodes in the tree
for (String[] row : lines) {
for (String line : row) {
        if (line != null) System.out.print(line + " ");
}
System.out.print("\n");
}
System.out.print("\n");
}

public static void createTree1(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.

// First the leaves
BinaryTree<String> hTree = new BinaryTree<>("Is it an android phone?");
BinaryTree<String> iTree = new BinaryTree<>("Is it a desktop?");
BinaryTree<String> jTree = new BinaryTree<>("Is it a drill?");
BinaryTree<String> kTree = new BinaryTree<>("Is it a chisel?");
BinaryTree<String> lTree = new BinaryTree<>("Is it a penguin?");
BinaryTree<String> mTree = new BinaryTree<>("Is it a camel?");
BinaryTree<String> nTree = new BinaryTree<>("Is it a goldfish?");
BinaryTree<String> oTree = new BinaryTree<>("Is it a dog?");

// Now the subtrees joining leaves:
        BinaryTree<String> dTree = new BinaryTree<>("Is it a bird?", lTree, mTree);
        BinaryTree<String> eTree = new BinaryTree<>("Is it a fish?", nTree, oTree);
        BinaryTree<String> fTree = new BinaryTree<>("Does it have a touch screen?", hTree,
iTree);
        BinaryTree<String> gTree = new BinaryTree<>("Is it a tool?", jTree, kTree);

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

        // Now the root
        tree.setTree("Are you thinking of an animal?", bTree, cTree);
        } // end createTree1

        public static void question(BinaryTree<String> tree) {
        // Looping over until loop is broken by System.exit or an option is chosen when a leaf
node is reached 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 = getInput(currentNode.getData());
                // Comparing user input and leading them down different node children
depending on their answer
                if (ans.equals("yes")) {
                currentNode = currentNode.getLeftChild();
                } else if (ans.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 = getInput(currentNode.getData());
        // Taking input and determining if yes or no
        // Leading them to the next function depending on their input e.g. exiting, saving, loading
        if (ans.equals("yes")) {
                JOptionPane.showMessageDialog(null, "The tree guessed correctly!\n");
                ans = 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 (ans.equals("1")) currentNode = tree.getRootNode();
                if (ans.equals("2")) {
                FileIO.storeTree(tree);
                currentNode = tree.getRootNode();
                }
                if (ans.equals("3")) {
                tree = FileIO.loadTree();
                currentNode = tree.getRootNode();
                }
                if (ans.equals("4")) System.exit(0);
        }
        // Adding node of information to tree on instance where tree is wrong
        // This involves setting the new correct answer and question for the tree's current answer
to reflect being wrong
        else if (ans.equals("no")){
                ans = getInput("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 = getInput("Distinguishing question?\n");
                currentNode.setData(ans);
                currentNode = tree.getRootNode();
        } else {
                JOptionPane.showMessageDialog(null, "Please enter yes or no.\n");
        }
        }
        }

        public static String getInput(String question) {
        String in = JOptionPane.showInputDialog(null, question);
        if (in == null) System.exit(0);
        else if (in.equals("")){
        JOptionPane.showMessageDialog(null, "Please enter a valid input");
        return getInput(question);
        } else {
        return in;
        }
        return in;
        }

        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

}

// BinaryTree.java
/*
 * A class that implements the ADT binary tree.
 *
 * @author Frank M. Carrano, Improved upon by Conor Mc Govern
 * @version 2.0
 */

public class BinaryTree<T> implements BinaryTreeInterface<T>, java.io.Serializable
{
        private static final long serialVersionUID = -1932834476575953631L;
        private BinaryNodeInterface<T> root;

        public BinaryTree()
        {
        root = null;
        } // end default constructor

        public BinaryTree(T rootData)
        {
        root = new BinaryNode<>(rootData);
        } // end constructor

        public BinaryTree(T rootData, BinaryTree<T> leftTree,
                BinaryTree<T> rightTree)
        {
        privateSetTree(rootData, leftTree, rightTree);
        } // end constructor

        public void setTree(T rootData)
        {
```

```
root = new BinaryNode<>(rootData);
} // end setTree

public void setTree(T rootData, BinaryTreeInterface<T> leftTree,
          BinaryTreeInterface<T> rightTree)
{
privateSetTree(rootData, (BinaryTree<T>)leftTree,
          (BinaryTree<T>)rightTree);
} // end setTree

// 26.08
private void privateSetTree(T rootData, BinaryTree<T> leftTree,
                    BinaryTree<T> rightTree)
{
root = new BinaryNode<>(rootData);

if ((leftTree != null) && !leftTree.isEmpty())
root.setLeftChild(leftTree.root);

if ((rightTree != null) && !rightTree.isEmpty())
{
if (rightTree != leftTree)
          root.setRightChild(rightTree.root);
else
          root.setRightChild(rightTree.root.copy());
} // end if

if ((leftTree != null) && (leftTree != this))
leftTree.clear();

if ((rightTree != null) && (rightTree != this))
rightTree.clear();
} // end privateSetTree

// 26.09
public T getRootData()
{
T rootData = null;

if (root != null)
rootData = root.getData();

return rootData;
```

```
} // end getRootData

// 26.09
public boolean isEmpty()
{
return root == null;
} // end isEmpty

// 26.09
public void clear()
{
root = null;
} // end clear

// 26.09 Not used
protected void setRootData(T rootData)
{
root.setData(rootData);
} // end setRootData

// 26.09 Not used
protected void setRootNode(BinaryNodeInterface<T> rootNode)
{
root = rootNode;
} // end setRootNode

// 26.09
protected BinaryNodeInterface<T> getRootNode()
{
return root;
} // end getRootNode

// 26.10
public int getHeight()
{
return root.getHeight();
} // end getHeight

// 26.10
public int getNumberOfNodes()
{
return root.getNumberOfNodes();
} // end getNumberOfNodes
```

```
	// Here down was amended by Conor Mc Govern
	public String[][] inorderTraverse()
	{
	// Creating an array that is root.getHeight deep and 2 x root.getHeight wide, as for every
time it gets deeper, it is twice as wide
	String[][] lines = new String[root.getHeight()][root.getHeight() * root.getHeight()];
	inorderTraverse(root, lines);
	return lines;
	} // end inorderTraverse

	private void inorderTraverse(BinaryNodeInterface<T> node, String[][] lines)
	{
	if (node != null)
	{
	inorderTraverse(node.getLeftChild(), lines);
	int i = 0;
	// Adding to array upside down depending on their height.
	while (true) {
		if (lines[node.getHeight() - 1][i] == null) {
		lines[node.getHeight() - 1][i] = node.getData().toString();
		break;
		}
		i++;
	}
	inorderTraverse(node.getRightChild(), lines);
	} // end if
	} // end inorderTraverse

} // end BinaryTree
```

FileIO.java

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

/* CT2109 NUI Galway:
 * Class to handle file io in project
 * @author Conor Mc Govern
 */

public class FileIO extends JPanel {
	Scanner s = new Scanner(System.in);
```

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

        public static void storeTree(BinaryTree<String> tree) {
        String question = "Please enter the name of the time you would like to store to under
path: " + System.getProperty("user.dir");
        // Serialization
        try
        {
        //Saving of object in a file
        FileOutputStream file = new FileOutputStream(getInput(question));
        ObjectOutputStream out = new ObjectOutputStream(file);

        // Method for serialization of object
        out.writeObject(tree);

        out.close();
        file.close();
        }

        catch(IOException ex)
        {
        JOptionPane.showMessageDialog(null, "Please enter a filename");
        System.out.println("IOException is caught");
        storeTree(tree);
        }
        }

        public static BinaryTree<String> loadTree() {
        BinaryTree<String> tree = null;
        String question = "Please enter the name of the time you would like to load to under
path: " + System.getProperty("user.dir");
        // Deserialization
```

```
	try {
	// Reading the object from a file
	FileInputStream file = new FileInputStream(getInput(question));
	ObjectInputStream in = new ObjectInputStream(file);

	// Method for deserialization of object
	tree = (BinaryTree<String>) in.readObject();

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

	return tree;
	}

	catch(IOException ex)
	{
	JOptionPane.showMessageDialog(null, "Please enter a filename");
	System.out.println("IOException is caught");
	return loadTree();

	}

	catch(ClassNotFoundException ex)
	{
	JOptionPane.showMessageDialog(null, "Please enter a filename");
	System.out.println("ClassNotFoundException is caught");
	return loadTree();
	}
	}
}
```

BinaryNode.java

```
/**
 * A class that represents nodes in a binary tree.
 *
 * @author Frank M. Carrano
 * @version 2.0
 */
class BinaryNode<T> implements BinaryNodeInterface<T>, java.io.Serializable
{
	private static final long serialVersionUID = 6828929352995534482L; // needed for
serializable objects
```

```
        private T data;
        private BinaryNode<T> left;
        private BinaryNode<T> right;

        public BinaryNode()
        {
        this(null); // call next constructor
        } // end default constructor

        public BinaryNode(T dataPortion)
        {
        this(dataPortion, null, null); // call next constructor
        } // end constructor

        public BinaryNode(T dataPortion, BinaryNode<T> leftChild, BinaryNode<T> rightChild) {
        data = dataPortion;
        left = leftChild;
        right = rightChild;
        } // end constructor


        public T getData() { return data; } // end getData

        public void setData(T newData) { data = newData; } // end setData

        public BinaryNodeInterface<T> getLeftChild() { return left; } // end getLeftChild

        public BinaryNodeInterface<T> getRightChild() { return right; } // end getRightChild

        public void setLeftChild(BinaryNodeInterface<T> leftChild) { left =
(BinaryNode<T>)leftChild; } // end setLeftChild

        public void setRightChild(BinaryNodeInterface<T> rightChild) { right =
(BinaryNode<T>)rightChild; } // end setRightChild

        public boolean hasLeftChild()
        {
        return left != null;
        } // end hasLeftChild

        public boolean hasRightChild()
        {
        return right != null;
```

```
} // end hasRightChild

public boolean isLeaf()
{
return (left == null) && (right == null);
} // end isLeaf

// 26.06
public BinaryNodeInterface<T> copy()
{
BinaryNode<T> newRoot = new BinaryNode<T>(data);

if (left != null)
newRoot.left = (BinaryNode<T>)left.copy();

if (right != null)
newRoot.right = (BinaryNode<T>)right.copy();

return newRoot;
} // end copy

// 26.11
public int getHeight()
{
return getHeight(this); // call private getHeight
} // end getHeight

// 26.11
private int getHeight(BinaryNode<T> node)
{
int height = 0;

if (node != null)
height = 1 + Math.max(getHeight(node.left), getHeight(node.right));

return height;
} // end getHeight

// 26.11
public int getNumberOfNodes()
{
int leftNumber = 0;
int rightNumber = 0;
```

```
		if (left != null)
		leftNumber = left.getNumberOfNodes();

		if (right != null)
		rightNumber = right.getNumberOfNodes();

		return 1 + leftNumber + rightNumber;
		} // end getNumberOfNodes
} // end BinaryNode
```

BinaryNodeInterface.java

```
/**
 * An interface for a node in a binary tree.
 *
 * @author Frank M. Carrano
 * @version 2.0
 */
interface BinaryNodeInterface<T>
{
		/** Task: Retrieves the data portion of the node.
		*  @return the object in the data portion of the node */
		public T getData();

		/** Task: Sets the data portion of the node.
		*  @param newData  the data object */
		public void setData(T newData);

		/** Task: Retrieves the left child of the node.
		*  @return the node that is this nodeÕs left child */
		public BinaryNodeInterface<T> getLeftChild();

		/** Task: Retrieves the right child of the node.
		*  @return the node that is this nodeÕs right child */
		public BinaryNodeInterface<T> getRightChild();

		/** Task: Sets the nodeÕs left child to a given node.
		*  @param leftChild  a node that will be the left child */
		public void setLeftChild(BinaryNodeInterface<T> leftChild);

		/** Task: Sets the nodeÕs right child to a given node.
		*  @param rightChild  a node that will be the right child */
		public void setRightChild(BinaryNodeInterface<T> rightChild);
```

```
        /** Task: Detects whether the node has a left child.
        *  @return true if the node has a left child */
        public boolean hasLeftChild();

        /** Task: Detects whether the node has a right child.
        *  @return true if the node has a right child */
        public boolean hasRightChild();

        /** Task: Detects whether the node is a leaf.
        *  @return true if the node is a leaf */
        public boolean isLeaf();

        /** Task: Counts the nodes in the subtree rooted at this node.
        *  @return the number of nodes in the subtree rooted at this node */
        public int getNumberOfNodes();

        /** Task: Computes the height of the subtree rooted at this node.
        *  @return the height of the subtree rooted at this node */
        public int getHeight();

        /** Task: Copies the subtree rooted at this node.
        *  @return the root of a copy of the subtree rooted at this node */
        public BinaryNodeInterface<T> copy();
} // end BinaryNodeInterface
```

BinaryTreeInterface.java

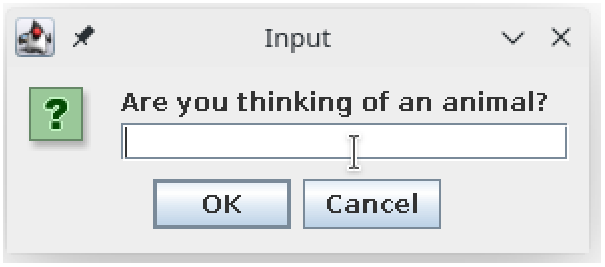
```
/**
 * An interface for the ADT binary tree.
 *
 * @author Frank M. Carrano
 * @version 2.0
 */
public interface BinaryTreeInterface<T> extends TreeInterface<T>
{
        /** Task: Sets an existing binary tree to a new one-node binary tree.
        *  @param rootData   an object that is the data in the new treeÕs root
        */
        public void setTree(T rootData);

        /** Task: Sets an existing binary tree to a new binary tree.
        *  @param rootData   an object that is the data in the new treeÕs root
        *  @param leftTree   the left subtree of the new tree
        *  @param rightTree  the right subtree of the new tree */
        public void setTree(T rootData, BinaryTreeInterface<T> leftTree,
                BinaryTreeInterface<T> rightTree);
} // end BinaryTreeInterface
```

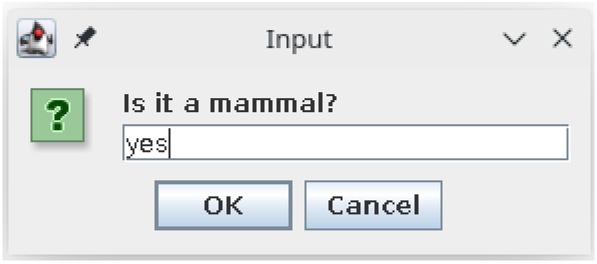
TreeInterface.java

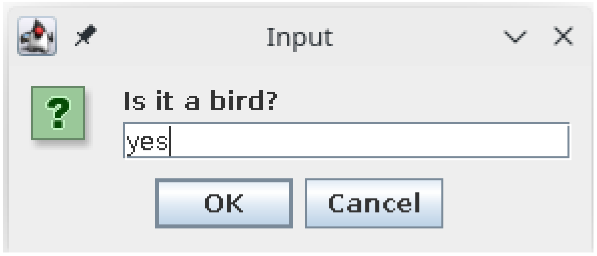
```
/**
 * An interface for basic operations of a tree.
 *
 * @author Frank M. Carrano
 * @version 2.0
 */
public interface TreeInterface<T>
{
        public T getRootData();
        public int getHeight();
        public int getNumberOfNodes();
        public boolean isEmpty();
        public void clear();
} // end TreeInterface
```

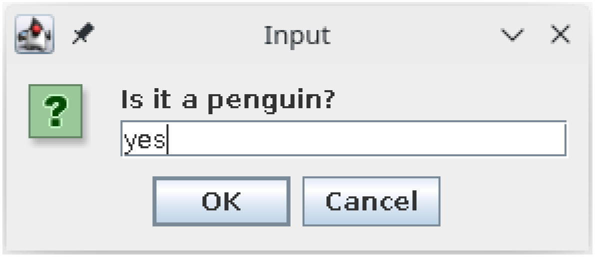
Testing

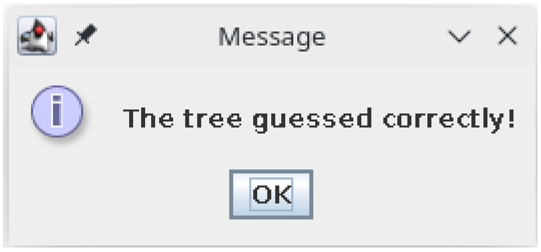
(Read pictures left to right)


Are you thinking of an animal?

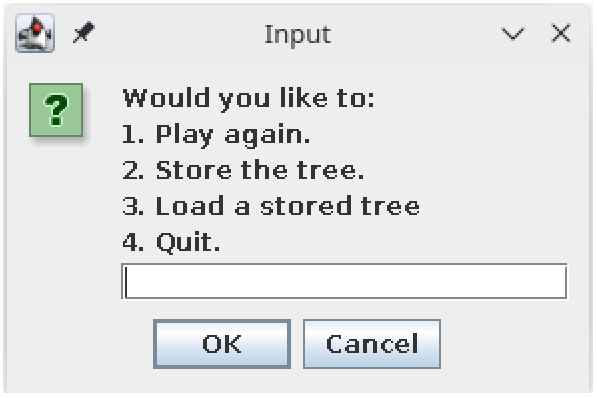

Is it a mammal?
yes


Is it a bird?
yes


Is it a penguin?
yes

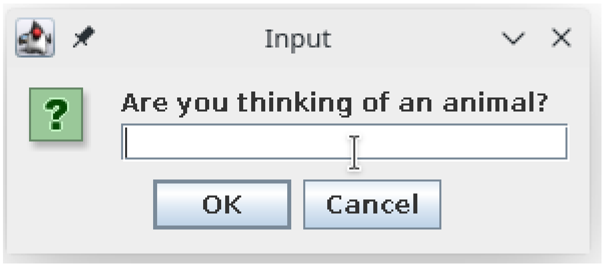

Message

The tree guessed correctly!

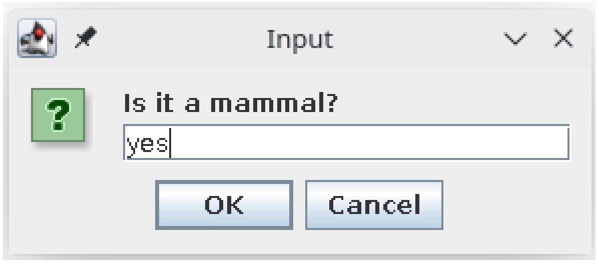

Input

Would you like to:
1. Play again.
2. Store the tree.
3. Load a stored tree
4. Quit.

Normal situation

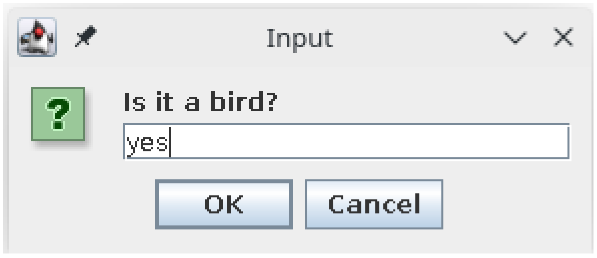

Input

Are you thinking of an animal?


Input

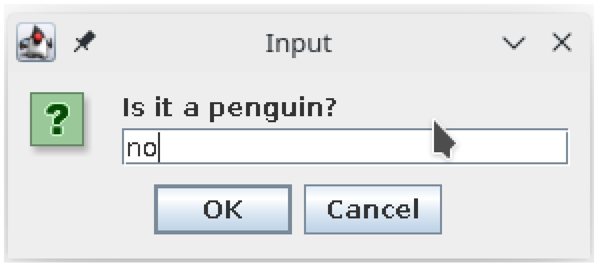
Is it a mammal?

yes


Input

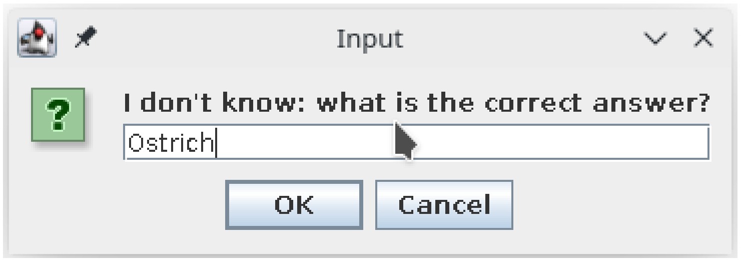
Is it a bird?

yes


Input

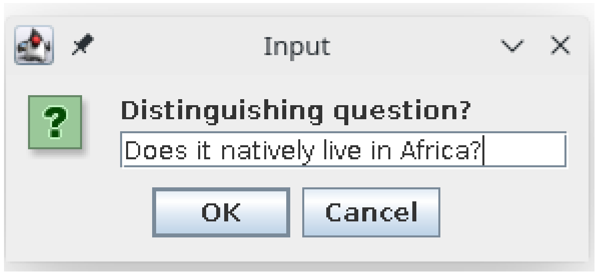
Is it a penguin?

no


Input

I don't know: what is the correct answer?

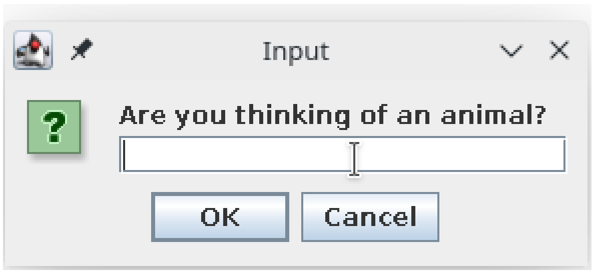
Ostrich


Input

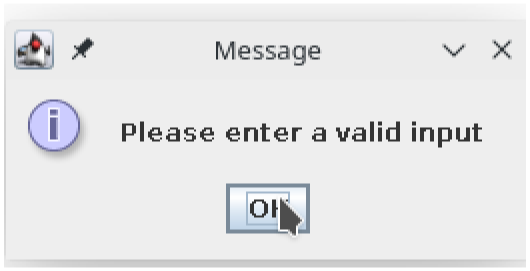
Distinguishing question?

Does it natively live in Africa?

Then the code loop again to reflect the amended node.

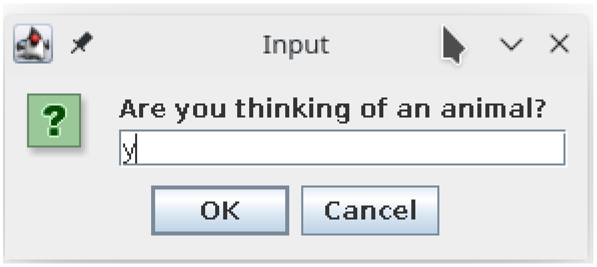

Input

Are you thinking of an animal?


Message

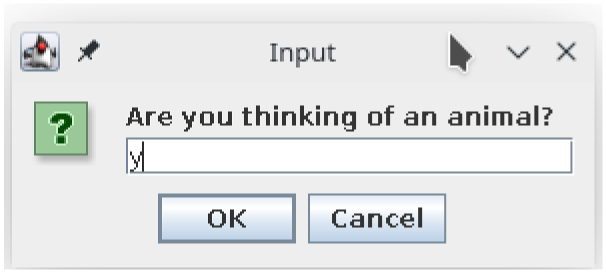
Please enter a valid input

When you input nothing, you get an error to input something


Input

Are you thinking of an animal?

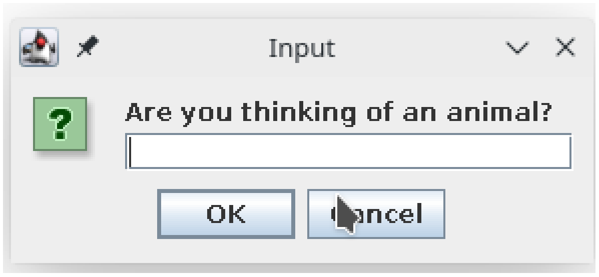
y


Input

Are you thinking of an animal?

y

When you input something but its not correct, it tells you to input yes or no


Input

Are you thinking of an animal?

When you click cancel (input == null), it exits.

When I press 2 at the end, I am able to serialize the object and see the file.
When I press 3, it is able to deserialize the object from the file.