CS 1150 Design Notebook Required Sections

Step 1: Problem Statement

This assignment will have me read a file and create parrot-like objects and use those parrots to fill a binary tree. The binary tree will be created from a class and will have specific methods to take in parrot objects. It will use a node with life and right pointers and will be organized by separating small and big values into left and right sides. From the tree, the code will display a hidden song by going through the tree in level order and after will display only the leaf nodes.

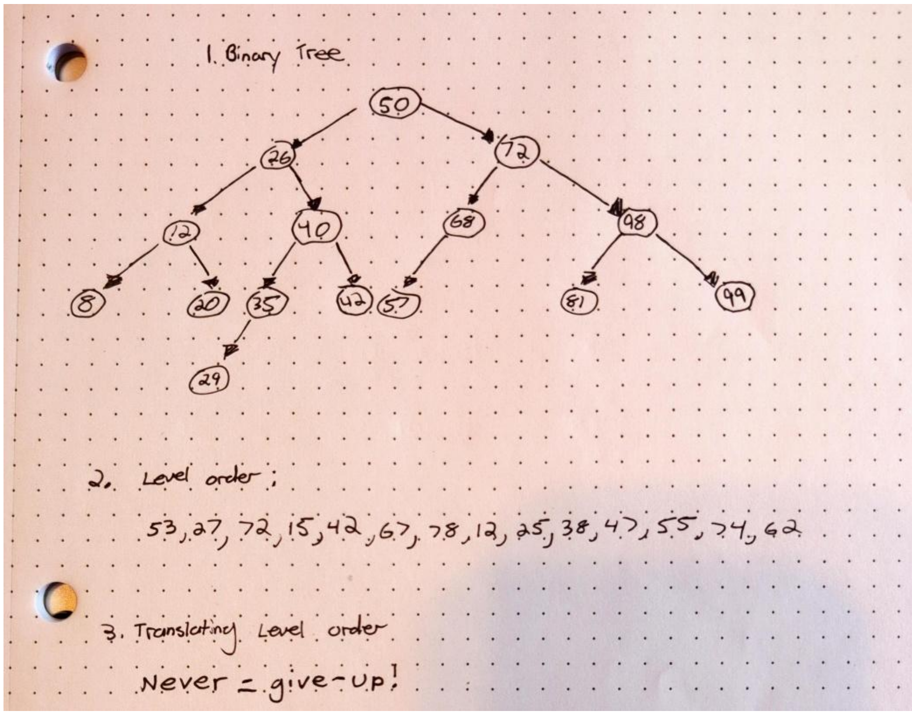
Step 2: Understandings

- What I Know:
  - Method
  - Objects
  - Comparable Interface
- What I Don't Know:
  - Creating a Binary Tree

Step 3: Pseudocode and pictures

Main:

- Create BinaryTree object
  - new BinnaryTree()
- Create File and Scanner for File
  - File file = new File(parrot.txt)
  - Scanner fileRead = new Scanner(file)
- Go through file and create parrot objects
  - Use while and hasNext()
- Call levelOrder Method in BinaryTree Class
- Call visitLeaves Method In BinaryTree Class



2. Level order:

53, 27, 72, 15, 42, 67, 78, 12, 25, 38, 47, 55, 74, 62

3. Translating Level order

Never-give-up!

Step 4: Lesson Learned

I was having an issue with creating parrot objects because it would get stuck creating parrot objects. I realized it was because I had my if statement checking for > or < 1 and but the id for the parrot was 93 and the head node id was 94 so the returned value was one so it did not go into either if statement, I changed them to be < or > 0 and also added a else statement for the possibility of equal values.

Step 5: Code

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

/*
Isaiah Hoffer
CS1450 (M/W)
5/1/25
Assignment 10
This assignment will make me create my own binary tree and fill its nodes with parrot objects.
It will use java's comparable interface to organize tree. Parrot objects are created from the given file.
*/
public class HofferIsaiahAssignment10 {

        public static void main(String[] args) throws FileNotFoundException {

                //Creating BinaryTree
                BinaryTree parrotTree = new BinaryTree();

                //Creating And Scanning File
                File parrotFile = new File("parrots.txt");
                Scanner parrotRead = new Scanner(parrotFile);

                //Read FIle Until No More Text
                while(parrotRead.hasNext()) {

                        int parrotID = parrotRead.nextInt(); //Gets ID Of Parrot
                        String parrotName = parrotRead.next(); //Gets Name Of Parrot
                        String parrotLyric = parrotRead.nextLine().trim(); //Gets Parrot Song Word

                        //Creating And Insterting Parrot To Binary Tree
                        parrotTree.insert(new Parrot(parrotID, parrotName, parrotLyric));
                }//While

                //Caling LevelOrder Method
                //Pretext
                System.out.printf("Parrot's Song\n"
                                + "------------------------------------------\n");
                parrotTree.levelOrder();

                //Calling visitLeaves Method
        //Pretext
        System.out.printf("\n\nParrots On Leaf Nodes\n"
                + "---------------------------------------\n");
        parrotTree.visitLeaves();

        //Closing Scanner
        parrotRead.close();

    }//main
}//class

//Parrot Object To Hold Id, Name, And Song Lyric
class Parrot implements Comparable<Parrot> {

    //Private Data
    private int id; //Holds Parrot ID
    private String name; //Holds Parrots Name
    private String songWord; //Hold Parrto's Lyric For Song

    public Parrot(int id, String name, String songWord) {


        //Initalizing Data
        this.id = id;
        this.name = name;
        this.songWord = songWord;

    }//Parrot Constructor

    //Getter To Return Name
    public String getName() {

        return name;
    }//GetName Method

    //Getter For SongWord
    public String sing() {

        return songWord;
    }//Sing Method

    @Override
    public int compareTo(Parrot otherParrot) {

        return this.id - otherParrot.id;

    }//compareTo Method

}//Parrot Class

class BinaryTree {
```

```
//Private Data
TreeNode root; //Head Of Tree

public BinaryTree() {

    //Initalizing Private Data
    this.root = null;
}//BinaryTree Constructor

//Adds Parrot To Tree, Returns True Or False
public boolean insert(Parrot parrotToAdd) {

    //Creating Node
    TreeNode treeNode = new TreeNode(parrotToAdd);

    //Boolean Variable
    Boolean didInsert = false;

    if(root == null) {
        root = treeNode;
    }//If

    else {

        TreeNode current = root;

        while(!didInsert) {

            //Left Side
            if(parrotToAdd.compareTo(current.parrot) < 0) {

                if(current.left == null) {

                    current.left = treeNode;
                    didInsert = true;
                }//If
                else {
                    current = current.left;
                }//Else
            }//If
            //Right Side
            else if(parrotToAdd.compareTo(current.parrot) > 0) {

                if(current.right == null) {

                    current.right = treeNode;
                    didInsert = true;
                }//If
                else {
                    current = current.right;
                }//Else

            }//Else If
            //Do Not Insert --- Duplicate Value
            else didInsert = false; //Else

        }//While
    }//Else

    return didInsert;
}//Insert Method

//Traverse Tree, Prints Parrot's Lyric
public void levelOrder() {

    if(root != null) {

        //Creating Queue
        Queue<TreeNode> nodeQueue = new LinkedList<>();

        //Adding Root To Queue
        nodeQueue.offer(root);

        while(!nodeQueue.isEmpty()) {

            TreeNode current = nodeQueue.remove();
            System.out.printf("%s ",current.parrot.sing());

            //Adding Left Node
            if(current.left != null) {

                nodeQueue.offer(current.left);
            }//If

            //Adding Right Node
            if(current.right != null) {

                nodeQueue.offer(current.right);
            }//If

        }//While
    }//If
}//levelOrder Method

//Calls Private visitLeaves Class
public void visitLeaves() {

    visitLeaves(this.root);
}//VisitLeaves Public Method

//Recusive Method To Display Only Leaves
```

```
    private void visitLeaves(TreeNode aNode) {

        //Only Traverse Filled Nodes
        if(aNode != null) {

            visitLeaves(aNode.left); //Goes All Left
            if(aNode.left == null && aNode.right == null) { //Checks If Node Is Leaf
                System.out.println(aNode.parrot.getName());
            }
            visitLeaves(aNode.right); //Go To Right As Climbing Back Up

        }//While

    }//VisitLeaves Private Method

    private class TreeNode {

        //Private Data
        Parrot parrot; //Hold Parrot Object
        TreeNode left; //Pointer To Go Left Of Tree
        TreeNode right; //Pointer To Go Right Of Tree

        public TreeNode(Parrot parrot) {

            //Initalizing Private Data
            this.parrot = parrot;
            this.left = null;
            this.right = null;
        }//TreeNode Constructor

    }//TreeNode Private Class

}//BinaryTree Class
```