Isaac Ray Shoebottom
CS 1083 (FR02B)
Assignment 11
3429069

Source Code:

NameCountTree.java:

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

/**
 * Class that represents a binary tree of names sorted alphabetically
 * @author Isaac Shoebottom (3429069)
 */

public class NameCountTree {
    /**
     * Root node, one at the top of the binary tree
     */
    private Node root;

    /**
     * Public method for initiating addition to the tree
     * @param valueIn String which represents the name
     */
    public void add(String valueIn) {
        if (root == null) {
            root = new Node(new Pair(valueIn));
        }
        else {
            add(valueIn, root);
        }
    }
```

```
    /**
     * Private method that recursively adds to the tree
     * @param valueIn String that represents the name
     * @param root The node that should be the new root for searching
where to add the new node
     */
    private void add(String valueIn, Node root) {
        if (valueIn.equalsIgnoreCase(root.info.name)) {
            root.info.count++;
        }
        else if (valueIn.compareToIgnoreCase(root.info.name) < 0) {
            if (root.left == null) {
                root.left = new Node(new Pair(valueIn));
            }
            else {
                add(valueIn, root.left);
            }
        }
        else {
            if (root.right == null) {
                root.right = new Node(new Pair(valueIn));
            }
            else {
                add(valueIn, root.right);
            }
        }

    }

    /**
```

```
     * Method for reading the input from a file
     * @param filename String The name of the file to be read
     * @throws IOException If the file does not exist
     */
    public void readText(String filename) throws IOException {
        Scanner sc = new Scanner(new File(filename));

        while (sc.hasNext()) {
            add(sc.next().toLowerCase());
        }
    }

    /**
     * Method to initiate a print of the tree
     */
    public void print() {
        print(root);
    }

    /**
     * Private method that recursively prints the tree. Relies on that
the tree is sorted alphabetically already
     * @param root  The node that is to be treated as root
     */
    private void print(Node root) {
        if (root != null) {
            print(root.left);
            System.out.println(root.info.toString());
            print(root.right);
        }
```

```
    }


    /**
     * Method to print the smallest node alphabetically
     */
    public void printMin() {
        //in case of empty file
        if (root != null) {
            Node temp = root;
            while (temp.left != null) {
                temp = temp.left;
            }


            System.out.print(temp.info.toString());
        }
    }


    /**
     * Node class represents a node containing the left right and pair
of information
     */
    static class Node {
        /**
         * The left node to current node
         */
        private Node left;
        /**
         * The right node to current node
         */
        private Node right;
```

```
        /**
         * The info in the current node
         */
        private final Pair info;

        /**
         * The constructor for a node
         * @param info Takes in a pair and constructs a node out of it
         */
        Node (Pair info) {
            this.info = info;
        }
    }

    /**
     * Class that represents a pair of data, the name and count of how
many of that name in the tree
     */
    static private class Pair {
        /**
         * The name in the pair
         */
        private final String name;
        /**
         * The count of how many of that name are in the tree
         */
        private int count = 1;

        /**
         * The constructor for the pair
```

```
         * @param name String of the name of the pair
         */
        Pair(String name) {
            this.name = name;
        }

        /**
         * Reimplementation of toString to not reference the memory
pointer but the contents of the class
         * @return The string containing the name and count
         */
        public String toString(){
            return "(" + name + ", " + count + ")";
        }
    }
}
```

Inputs

Input 1:

```
Elizabeth Anne Flynn

Sarah Blair

Abe Corey

Blair Elliot

COREY FLYNN

anne blair
```

Input 2:

```
Elizabeth
```

Input 3:

```
Elizabeth Anne Flynn

Sarah Blair

Abe Corey

Blair Elliot

COREY FLYNN

anne blair

isaac ray avery

spencer adam

genna

Abcde
```

Outputs

Output 1:

```
C:\Programs\Java\8\bin\java.exe ...
(abe, 1)
(anne, 2)
(blair, 3)
(corey, 2)
(elizabeth, 1)
(elliot, 1)
(flynn, 2)
(sarah, 1)
Minimum entry in tree:
(abe, 1)
Process finished with exit code 0
```

Output 2:

```
C:\Programs\Java\8\bin\java.exe ...
(elizabeth, 1)
Minimum entry in tree:
(elizabeth, 1)
Process finished with exit code 0
```

Output 3:

```
C:\Programs\Java\8\bin\java.exe ...
(abcde, 1)
(abe, 1)
(adam, 1)
(anne, 2)
(avery, 1)
(blair, 3)
(corey, 2)
(elizabeth, 1)
(elliot, 1)
(flynn, 2)
(genna, 1)
(isaac, 1)
(ray, 1)
(sarah, 1)
(spencer, 1)
Minimum entry in tree:
(abcde, 1)
Process finished with exit code 0
```