

New York Institute of Technology

Music File Organizer

Michael Brower & Vitor Cavalcante

CSCI 260 W02

Dr. Michael Nizich

12-17-19

## **Student Contributions Page**

Vitor programmed the majority of the project. He programmed the various data structures so they can efficiently organize music samples supplied by the user of the program.

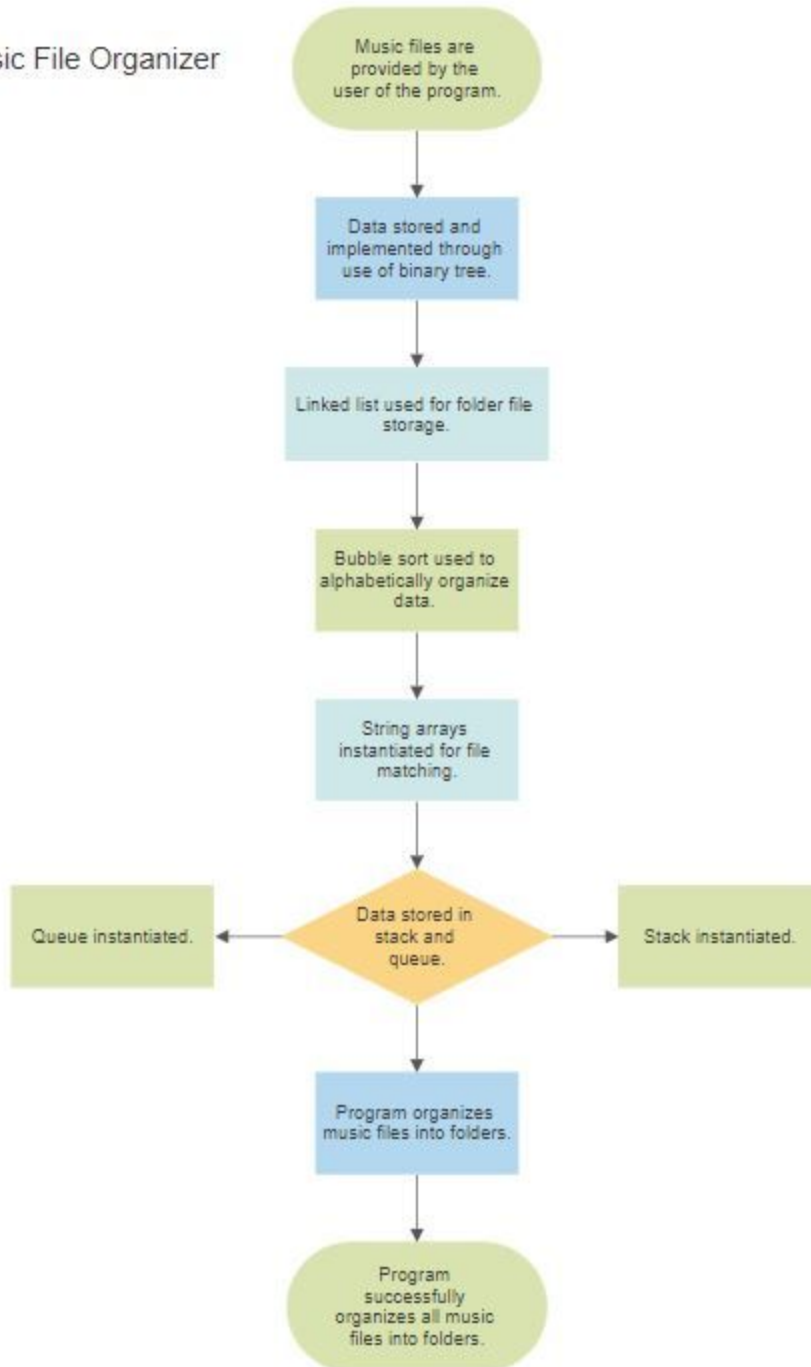
Michael prepared the majority of the report. He provided the flow chart and project summary along with other things.

## **Problem Statement**

This program takes music files provided by the user of the program and organizes them using a variety of data structures. When someone is using these music samples to create a beat, they can become disorganized. This program organizes all the samples provided into folders so they can be easily found and used. This program would be useful to a music producer or anyone who works with music samples.

## Project Flow Chart

### Music File Organizer



## Screenshots

```
/*With the modified BT and BTNode class so that it accepts String values and stores it in the tree in a
 * alphabetical manner, I am placing the  from the folder into the binary tree as it read the files from the
 * folder. */
BT bt = new BT();

for(File file : musicFolder.listFiles())
{
    bt.insert(file.getName());
}

System.out.println("Successfully stored in the Binary Tree Alphabetically \n");

private String headNodeData;

/* Functions to insert data */
public void insert(String data)
{
    root = insert(root, data);
    headNodeData = data;
}

/* Function to insert data recursively */ //Check if less or equal to or greater and organize the insert
private BTNode insert(BTNode node, String data)
{
    if (node == null)
    {
        node = new BTNode(data);
    }
    else
    {
        if ((node.getRight() == null) && ((data.compareTo(headNodeData) >= 0)))
            node.right = insert(node.right, data);
        else
            node.left = insert(node.left, data);
    }
    return node;
}
```

*Use of the binary tree and its implementation through the use of the insert method*

```
//store all files directories from the folder into an linkedList
LinkedList<File> musicFiles = new LinkedList<File>();

for(File file : musicFolder.listFiles())
{
    musicFiles.add(file);
}

System.out.println("Successfully stored in a Linked List \n");
```

### *Use of a Linked List for folder file storage*

```
//Creating an String array with all of the files in an alphabetically organized fashion
String[] sortedMusicFiles = musicFolder.list();
bubbleSort(sortedMusicFiles);

//for(String fileName: sortedMusicFiles)
//System.out.println("The Sorted Music Files is: " + fileName + "\n");

System.out.println("Successfully created an alphabetically organized instance of a String Array using BubbleSort \n");
```

```
public static void bubbleSort(String[] stringArray)
{
    int n = stringArray.length;
    String temp = null;

    for(int i=0; i<n; i++)
    {
        for(int j=1; j< (n-i); j++)
        {
            if(stringArray[j-1].compareTo(stringArray[j]) >= 0)
            {
                temp = stringArray[j-1];
                stringArray[j-1] = stringArray[j];
                stringArray[j] = temp;
            }
        }
    }
}
```

### *Use of a bubble sort and its method so it works with String values*

```
//create multiple String arrays to look for files that have one of the names in the array
// e.g. Scales (Abm, Cmaj, ... ), Instruments(guitar, bass, horns), Percussion (snare, hats,..)
String[] scales = {"Major", "Minor", "Dorian"};
String[] percussion = {"Snare", "Kick", "HiHat", "Cymbal"};
String[] loops = {"Loop"};
String[] instruments = {"Guitar", "Keys", "Bass", "Piano", "Conga", "Harmonica"};

System.out.println("Name checking arrays were instantiated \n");
```

### *String arrays instantiated with the names for the file matching*

```
//start comparing file names from the arraylist and the array with the titles for a match
//If matched
// 1. create a stack under the name of the array category and store the file in the stack
// 2. Put the file category name in a queue
Queue<String> folderCategoryName = new LinkedList<String>();

//Placing the file directories in the stack that correlates to its category type
// e.g. .\src\MusicFolder\AbMajor.txt into the scalesMatched Stack
Stack scalesMatched = new Stack();
Stack percussionMatched = new Stack();
Stack loopsMatched = new Stack();
Stack instrumentsMatched = new Stack();
Stack notMatched = new Stack();

System.out.println("Stacks and Queues created \n");
```

### *Stack and Queue Instantiation*

Problems Declaration Console

<terminated> FileOrganizer [Java Application] C:\Program Files\Java\jdk-10.0.2\bin\javaw.exe (Dec 8, 2019, 2:12:24 PM)

Successfully stored in the Binary Tree Alphabetically

Successfully stored in a Linked List

Successfully created an alphabetically organized instance of a String Array using BubbleSort

Name checking arrays were instantiated

Stacks and Queues created

Directory Scales was created successfully

Directory Instruments was created successfully

Directory Percussion was created successfully

Directory Loops was created successfully

Directory Unable to Categorize was created successfully

The File GMajor.txt was moved successfully

The File GbMajor.txt was moved successfully

The File FMajor.txt was moved successfully

The File F#Major.txt was moved successfully

The File EMajor.txt was moved successfully

The File EbMajor.txt was moved successfully

The File DMajor.txt was moved successfully

The File DbMajor.txt was moved successfully

The File CMajor.txt was moved successfully

The File CbMajor.txt was moved successfully

The File C#Major.txt was moved successfully

The File BMajor.txt was moved successfully

The File BbMajor.txt was moved successfully

The File AMajor.txt was moved successfully

The File AbMajor.txt was moved successfully

The File Snare2.txt was moved successfully

The File Snare1.txt was moved successfully

The File Snare.txt was moved successfully

The File Kick2.txt was moved successfully

The File Kick1.txt was moved successfully

The File Kick.txt was moved successfully

The File HiHat2.txt was moved successfully

The File HiHat1.txt was moved successfully

The File HiHat.txt was moved successfully

The File Cymbal2.txt was moved successfully

The File Cymbal1.txt was moved successfully

The File Cymbal.txt was moved successfully

The File PLoop2.txt was moved successfully

The File PLoop1.txt was moved successfully

The File PLoop.txt was moved successfully

The File GLoop2.txt was moved successfully

The File GLoop1.txt was moved successfully

The File GLoop.txt was moved successfully

The File Keys2.txt was moved successfully

The File Kevs1.txt was moved successfully

*Sample Output*








> Vitor Cavalcante > eclipse-workspace > CSCI260\_Project > src > MusicFolder

<input type="checkbox"/>	Name	Date modified	Type	Size
	AbMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
	AMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Bass.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Bass1.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Bass2.txt	12/7/2019 10:08 AM	Text Document	1 KB
- DC	BbMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
- Bip	BMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
- MC	C#Major.txt	12/7/2019 10:08 AM	Text Document	1 KB
	CbMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
	CMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Conga.txt	12/7/2019 10:08 AM	Text Document	0 KB
inal	Cymbal.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Cymbal1.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Cymbal2.txt	12/7/2019 10:08 AM	Text Document	1 KB
	DbMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
	DMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
	EbMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
	EMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
	F#Major.txt	12/7/2019 10:08 AM	Text Document	1 KB
Info	FMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
	GbMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
	GLoop.txt	12/7/2019 10:08 AM	Text Document	1 KB
	GLoop1.txt	12/7/2019 10:08 AM	Text Document	1 KB
	GLoop2.txt	12/7/2019 10:08 AM	Text Document	1 KB
	GMajor.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Guitar.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Guitar1.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Guitar2.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Harmonica.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Harmonica1.txt	12/7/2019 10:08 AM	Text Document	1 KB
	HiHat.txt	12/7/2019 10:08 AM	Text Document	1 KB
	HiHat1.txt	12/7/2019 10:08 AM	Text Document	1 KB
	HiHat2.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Keys.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Keys1.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Keys2.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Kick.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Kick1.txt	12/7/2019 10:08 AM	Text Document	1 KB
	Kirk2.txt	12/7/2019 10:08 AM	Text Document	1 KB

***Folder before organization taking place***

Vitor Cavalcante > eclipse-workspace > CSCI260\_Project > src > MusicFolder

<input type="checkbox"/>	Name	Date modified	Type	Size
	Instruments	12/8/2019 2:15 PM	File folder	
	Loops	12/8/2019 2:15 PM	File folder	
	Percussion	12/8/2019 2:15 PM	File folder	
	Scales	12/8/2019 2:15 PM	File folder	
	Unable to Categorize	12/8/2019 2:15 PM	File folder	

*Folder after organization taking place*

## Java Source Code

### FileCreator Class:

*//This class is simply a tool to create dummy data to use on the project*

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.List;

public class FileCreator
{

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

        //If I wanted to insert a string at the first and second line of the text file accordingly
        //List<String> lines = Arrays.asList("", "");

        //creates the file and names it
        //java.nio.file.Path file = Paths.get("Guitar2.txt");

        //writes in the file based on what you inserted on the variable lines
        //Files.write(file, lines, StandardCharsets.UTF_8);

        //List<String> lines = Arrays.asList("", "");
        //java.nio.file.Path file = Paths.get("C#Major.txt");
        //Files.write(file, lines, StandardCharsets.UTF_8);

        int loopCount = 2;
        String textFileName = "Harmonica";
        java.nio.file.Path file = null;
        List<String> lines = Arrays.asList("", "");

        for(int i = 0; i < loopCount; i++)
        {

            if (i == 0)
            {
                file = Paths.get(textFileName+".txt");
                Files.write(file, lines, StandardCharsets.UTF_8);
            }
            else
```

```

        {
            file = Paths.get(textFileName+i+".txt");
            Files.write(file, lines, StandardCharsets.UTF_8);
        }

    }

}

```

## FileOrganizer Class:

```

/* The focus of this class is to read all the files in the music folder, instantiate sub-folders within it,
 * and place all the files that correlate to that folder category inside the sub-folder.
 *
 * The MusicFolderOriginal folder consists of all of the original dummy data we are using for testing purposes
 * The MusicFolder is the folder we are using to organize the files and instantiate sub-folders in it in order to place
the files in their corresponding sub-folder
 *
 * AFTER EACH RUN DELETE ALL THE FILES FROM THE MusicFolder AND COPY THE FILES FROM
THE MusicFolderOriginal AND PLACE IT AT THE MusicFolder IN ORDER TO HAVE THE FileOrganizer
 * CLASS ORGANIZE THE FILES IN THE MusicFolder AGAIN.
 *
 */

```

```

import java.nio.file.Files;
import java.nio.file.Path;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;
import java.io.*;
import java.nio.file.*;

```

```

public class FileOrganizer
{

```

```

    //This is a bubble sort method to sort a string array and return the string array in a alphabetical manner.

```

```

    public static void bubbleSort(String[] stringArray)
    {
        int n = stringArray.length;
        String temp = null;

        for(int i=0; i<n; i++)
        {
            for(int j=1; j< (n-i); j++)
            {
                if(stringArray[j-1].compareTo(stringArray[j]) >= 0)
                {

```

```

        temp = stringArray[j-1];
        stringArray[j-1] = stringArray[j];
        stringArray[j] = temp;
    }
}
}

```

```

public static void main(String args[]) throws IOException
{
    //Creating a file for the folder where it contains all the musical dummy files
    File musicFolder = new File(".\\src\\MusicFolder");

    //File directory = new File(".");
    //System.out.println("The Path is: " + directory.getAbsolutePath());

    /*With the modified BT and BTNode class so that it accepts String values and stores it in the tree
    in a
    * alphabetical manner, I am placing the  from the folder into the binary tree as it read the files
    from the
    * folder. */
    BT bt = new BT();

    for(File file : musicFolder.listFiles())
    {
        bt.insert(file.getName());
    }

    System.out.println("Successfully stored in the Binary Tree Alphabetically \n");

    //store all files directories from the folder into an linkedList
    LinkedList<File> musicFiles = new LinkedList<File>();

    for(File file : musicFolder.listFiles())
    {
        musicFiles.add(file);
    }

    System.out.println("Successfully stored in a Linked List \n");

    //Creating an String array with all of the files in an alphabetically organized fashion
    String[] sortedMusicFiles = musicFolder.list();
    bubbleSort(sortedMusicFiles);

    //for(String fileName: sortedMusicFiles)
    //System.out.println("The Sorted Music Files is: " + fileName + "\n");

    System.out.println("Successfully created an alphabetically organized instance of a String Array
    using BubbleSort \n");
}

```

```

//System.out.println(musicFiles);

//create multiple String arrays to look for files that have one of the names in the array
// e.g. Scales (Abm, Cmaj, ... ), Instruments(guitar, bass, horns), Percussion (snare, hats,..)
String[] scales = {"Major", "Minor", "Dorian"};
String[] percussion = {"Snare", "Kick", "HiHat", "Cymbal"};
String[] loops = {"Loop"};
String[] instruments = {"Guitar", "Keys", "Bass", "Piano", "Conga", "Harmonica"};

System.out.println("Name checking arrays were instantiated \n");


//start comparing file names from the arraylist and the array with the titles for a match
//If matched
//      1. create a stack under the name of the array category and store the file in the stack
//      2. Put the file category name in a queue
Queue<String> folderCategoryName = new LinkedList<String>();

//Placing the file directories in the stack that correlates to its category type
// e.g. .\src\MusicFolder\AbMajor.txt into the scalesMatched Stack
Stack scalesMatched = new Stack();
Stack percussionMatched = new Stack();
Stack loopsMatched = new Stack();
Stack instrumentsMatched = new Stack();
Stack notMatched = new Stack();

System.out.println("Stacks and Queues created \n");

/* Using this flag to check if there was a match when comparing file name with the string arrays,
 * if there isn't one at the end of checking all string arrays then insert on the notMatched Stack and
Queue */
boolean match = false;

// Just to make sure I'm not inserting multiple instances of the category name in the queue so that I
can create only one folder of that type
boolean scalesQueueInserted = false;
boolean percussionQueueInserted = false;
boolean loopsQueueInserted = false;
boolean instrumentsQueueInserted = false;
boolean notMatchedQueueInserted = false;

File currentMusicFile;

//This while loops will both the stacks and Queue created previously, it will go through the entire
linkedList checking for the directory name and if there is a match
//then it stores it the respective stack where it falls under and adds the name of the category it falls
under into the queue so that the folder for that category can be
//instantiated later.
while(!musicFiles.isEmpty())
{

```

```

currentMusicFile = musicFiles.remove();

//System.out.println(currentMusicFile.getName());

for(String i : scales)
{
    //System.out.println(currentMusicFile.getName());
    //System.out.println("Checking for " + i + ": " +
currentMusicFile.getName().contains(i));

    if(currentMusicFile.getName().contains(i))
    {

        if(scalesQueueInserted == false)
        {
            folderCategoryName.add("Scales");
            scalesQueueInserted = true;
        }

        scalesMatched.add(currentMusicFile);
        match = true;
    }
}

for(String i : percussion)
{
    //System.out.println(currentMusicFile.getName());
    //System.out.println("Checking for " + i + ": " +
currentMusicFile.getName().contains(i));

    if(currentMusicFile.getName().contains(i))
    {
        if(percussionQueueInserted == false)
        {
            folderCategoryName.add("Percussion");
            percussionQueueInserted = true;
        }

        percussionMatched.add(currentMusicFile);
        match = true;
    }
}

for(String i : loops)
{
    //System.out.println(currentMusicFile.getName());
    //System.out.println("Checking for " + i + ": " +
currentMusicFile.getName().contains(i));

    if(currentMusicFile.getName().contains(i))
    {
        if(loopsQueueInserted == false)

```

```

        {
            folderCategoryName.add("Loops");
            loopsQueueInserted = true;
        }

        loopsMatched.add(currentMusicFile);
        match = true;
    }
}

for(String i : instruments)
{
    //System.out.println(currentMusicFile.getName());
    //System.out.println("Checking for " + i + ": " +
currentMusicFile.getName().contains(i));

    if(currentMusicFile.getName().contains(i))
    {
        if(instrumentsQueueInserted == false)
        {
            folderCategoryName.add("Instruments");
            instrumentsQueueInserted = true;
        }

        instrumentsMatched.add(currentMusicFile);
        match = true;
    }
}

if(match == false)
{
    if(notMatchedQueueInserted == false)
    {
        folderCategoryName.add("Unable to Categorize");
        notMatchedQueueInserted = true;
    }

    notMatched.add(currentMusicFile);
}

match = false;
}

```

```

//Checking if was placed correctly
/*
System.out.println("Category names are: ");
while(!folderCategoryName.isEmpty())
{

```



```

        System.out.println(folderCategoryName.remove());
    }

    System.out.println("\n");

    System.out.println("Scales Stack is: " + scalesMatched + "\n");
    System.out.println("percussion Stack is: " + percussionMatched + "\n");
    System.out.println("Loops Stack is: " + loopsMatched + "\n");
    System.out.println("Instruments Stack is: " + instrumentsMatched + "\n");
    System.out.println("NotMatched Stack is: " + notMatched + "\n");
    */

    //after analyzing the entire folder
    //    1. Instantiate new folders on the location of the folder we are analyzing and name it after
the categories that were matched and placed it in the queue (create while loop until queue is empty)
    //    2. empty the stack of that category in the folder

    //Creates the folder Directories with the category names being used to name the folders
    String folderName = null;

    while(!folderCategoryName.isEmpty())
    {
        folderName = folderCategoryName.peek();

        File file = new File(".\\src\\MusicFolder\\" + folderCategoryName.remove());

        boolean bool = file.mkdir();
        if(bool)
        {
            System.out.println("Directory " + folderName + " was created successfully");
        }
        else
        {
            System.out.println("Sorry couldn't create specified directory");
        }
    }

    System.out.println("\n");

    //Empties the Stack of Scales and moves the files into the respective "scales" folder
    while(!scalesMatched.isEmpty())
    {

        String textFileName = scalesMatched.pop().toString().substring(18);

        Path temp = Files.move
(Paths.get("C:\\Users\\vitor\\eclipse-workspace\\CSCI260_Project\\src\\MusicFolder\\" + textFileName),
Paths.get("C:\\Users\\vitor\\eclipse-workspace\\CSCI260_Project\\src\\MusicFolder\\Scales\\" + textFileName));

```

```

        if(temp != null)
        {
            System.out.println("The File " + textFileName + " was moved successfully");
        }
        else
        {
            System.out.println("Failed to move the file");
        }
    }

    //Empties the Stack of percussion and moves the files into the respective "percussion" folder
    while(!percussionMatched.isEmpty())
    {

        String textFileName = percussionMatched.pop().toString().substring(18);

        Path temp = Files.move
(Paths.get("C:\\Users\\vitor\\eclipse-workspace\\CSCI260_Project\\src\\MusicFolder\\" + textFileName),
Paths.get("C:\\Users\\vitor\\eclipse-workspace\\CSCI260_Project\\src\\MusicFolder\\percussion\\" + textFileName));

        if(temp != null)
        {
            System.out.println("The File " + textFileName + " was moved successfully");
        }
        else
        {
            System.out.println("Failed to move the file");
        }
    }

    //Empties the Stack of Loops and moves the files into the respective "Loops" folder
    while(!loopsMatched.isEmpty())
    {

        String textFileName = loopsMatched.pop().toString().substring(18);

        Path temp = Files.move
(Paths.get("C:\\Users\\vitor\\eclipse-workspace\\CSCI260_Project\\src\\MusicFolder\\" + textFileName),
Paths.get("C:\\Users\\vitor\\eclipse-workspace\\CSCI260_Project\\src\\MusicFolder\\Loops\\" + textFileName));

        if(temp != null)
        {
            System.out.println("The File " + textFileName + " was moved successfully");
        }
        else
        {
            System.out.println("Failed to move the file");
        }
    }
}

```

```

//Empties the Stack of Instruments and moves the files into the respective "Intruments" folder
while(!instrumentsMatched.isEmpty())
{

    String textFileName = instrumentsMatched.pop().toString().substring(18);

    Path temp = Files.move
(Paths.get("C:\\Users\\vitor\\eclipse-workspace\\CSCI260_Project\\src\\MusicFolder\\" + textFileName),
Paths.get("C:\\Users\\vitor\\eclipse-workspace\\CSCI260_Project\\src\\MusicFolder\\Instruments\\" +
textFileName));

    if(temp != null)
    {
        System.out.println("The File " + textFileName + " was moved successfully");
    }
    else
    {
        System.out.println("Failed to move the file");
    }
}

//Empties the Stack of notMatched and moves the files into the respective "Unable to Categorize"
folder
while(!notMatched.isEmpty())
{

    String textFileName = notMatched.pop().toString().substring(18);

    Path temp = Files.move
(Paths.get("C:\\Users\\vitor\\eclipse-workspace\\CSCI260_Project\\src\\MusicFolder\\" + textFileName),
Paths.get("C:\\Users\\vitor\\eclipse-workspace\\CSCI260_Project\\src\\MusicFolder\\Unable to Categorize\\" +
textFileName));

    if(temp != null)
    {
        System.out.println("The File " + textFileName + " was moved successfully");
    }
    else
    {
        System.out.println("Failed to move the file");
    }
}

}
}

```

## BTNode Class:

```
/* Class BTNode */
class BTNode
{
    BTNode left, right;
    String data;

    /* Constructor */
    public BTNode()
    {
        left = null;
        right = null;
        data = "Blank";
    }

    /* Constructor */
    public BTNode(String n)
    {
        left = null;
        right = null;
        data = n;
    }

    /* Function to set left node */
    public void setLeft(BTNode n)
    {
        left = n;
    }

    /* Function to set right node */
    public void setRight(BTNode n)
    {
        right = n;
    }

    /* Function to get left node */
    public BTNode getLeft()
    {
        return left;
    }

    /* Function to get right node */
    public BTNode getRight()
    {
        return right;
    }
}
```

```

/* Function to set data to node */
public void setData(String d)
{
    data = d;
}

/* Function to get data from node */
public String getData()
{
    return data;
}
}

/* Class BT */
class BT
{
    private BTNode root;

    /* Constructor */
    public BT()
    {
        root = null;
    }

    /* Function to check if tree is empty */
    public boolean isEmpty()
    {
        return root == null;
    }

    private String headNodeData;

    /* Functions to insert data */
    public void insert(String data)
    {
        root = insert(root, data);
        headNodeData = data;
    }

    /* Function to insert data recursively */ //Check if less or equal to or greater and organize the insert
    private BTNode insert(BTNode node, String data)
    {
        if (node == null)
        {
            node = new BTNode(data);

```

```

    }
    else
    {
        if ((node.getRight() == null) && ((data.compareTo(headNodeData) >= 0)))
            node.right = insert(node.right, data);
        else
            node.left = insert(node.left, data);
    }
    return node;
}

/* Function to count number of nodes */
public int countNodes()
{
    return countNodes(root);
}

/* Function to count number of nodes recursively */
private int countNodes(BTNode r)
{
    if (r == null)
        return 0;
    else
    {
        int l = 1;
        l += countNodes(r.getLeft());
        l += countNodes(r.getRight());
        return l;
    }
}

/* Function to search for an element */
public boolean search(String val)
{
    return search(root, val);
}

/* Function to search for an element recursively */
private boolean search(BTNode r, String val)
{
    if (r.equals(val))
        return true;
    if (r.getLeft() != null)
        if (search(r.getLeft(), val))
            return true;
    if (r.getRight() != null)
        if (search(r.getRight(), val))
            return true;
    return false;
}

```

```

/* Function for inorder traversal */
public void inorder()
{
    inorder(root);
}
private void inorder(BTNode r)
{
    if (r != null)
    {
        inorder(r.getLeft());
        System.out.print(r.getData() + " ");
        inorder(r.getRight());
    }
}

```

```

/* Function for preorder traversal */
public void preorder()
{
    preorder(root);
}

```

```

private void preorder(BTNode r)
{
    if (r != null)
    {
        System.out.print(r.getData() + " ");
        preorder(r.getLeft());
        preorder(r.getRight());
    }
}

```

```

/* Function for postorder traversal */
public void postorder()
{
    postorder(root);
}

```

```

private void postorder(BTNode r)
{
    if (r != null)
    {
        postorder(r.getLeft());
        postorder(r.getRight());
        System.out.print(r.getData() + " ");
    }
}

```

```

}

```

## **Requirement Documents**

- In order for the program to work two things are required from the user:
  - A folder where the user has all the music files stored in it where he wants it to be organized
  - The name specifications as to what the folder should be organized by.
    - e.g. Sub-folder named “Scales” where it stores all files under the name or contains “major” in it.



## **Project Summary**

The problem of having unorganized music files was solved by our program. Our program organizes music files into subfolders using various data structures. This program mainly helps to speed up the workflow of the user by allowing them to quickly find the music file they are looking for. This program would be mainly used by those who work with digital audio files.

## References

“Java - How to List .Txt File in a Directory.” *BORAJI.COM*,  
<https://www.boraji.com/java-how-to-list-txt-file-in-a-directory>.

Deranor, et al. “How to Create Multiple Files with the Terminal?” *Ask Ubuntu*, 1  
May 1965,  
<https://askubuntu.com/questions/605558/how-to-create-multiple-files-with-the-terminal>.

“Moving a File from One Directory to Another Using Java.” *GeeksforGeeks*, 19 May  
2017, <https://www.geeksforgeeks.org/moving-file-one-directory-another-using-java/>.