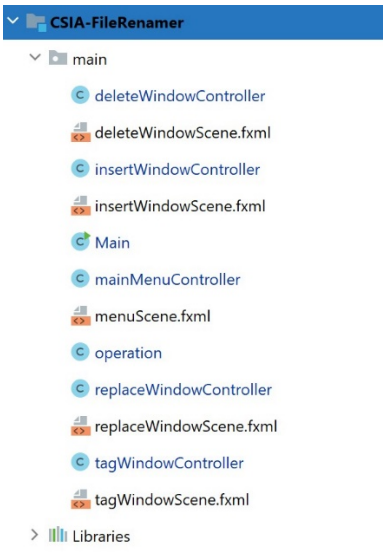
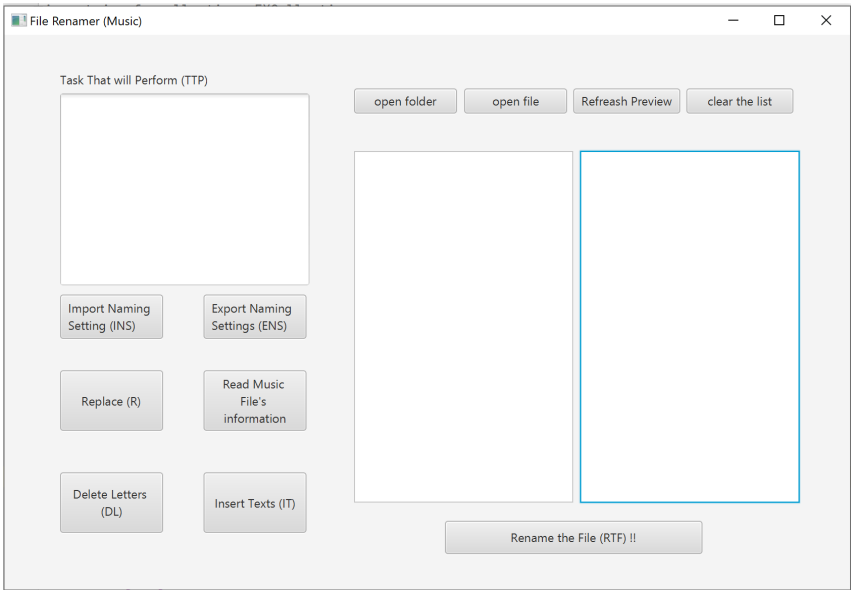
Criterion C: Development

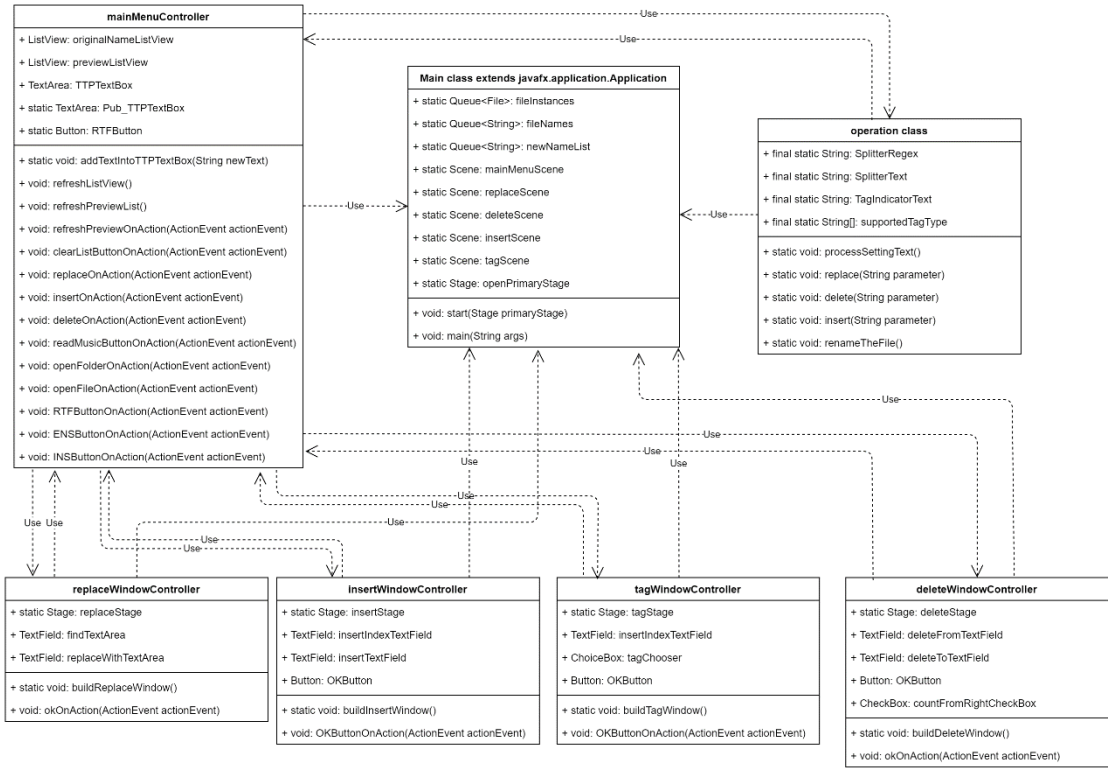
CSIA-FileRenamer is a Java-based program that helps the user to rename large amount of music files easily. Essentially, the user will interact with the GUI. They may push buttons and enter soe information to set rename rules. Whenever they finished the editing of a command through using GUI, the GUI will export the command in text into TTP (Task that will perform) TextBox onto GUI, and as the user pushes “refresh”, the program will process the commands in TTP TextBox and output the result on the preview name list, which will be apply to the real file after user click RTF (Rename the file) button. The program is using Javafx to build GUI interfaces, and therefore there are numerous layout files (.fxml) in the project, along with the controller classes that controls the behavior of the GUI interfaces.



Program’s structure



Main window



UML Class Diagram

There are 3 Queues implemented by Linked List in the Main class, which are being use throughout the program, as all the files, file names, and the result after renaming are stored in these 3 variables.

```
public class Main extends Application {

    //universal variables
    public static Queue<File> fileInstances = new LinkedList<>();
    public static Queue<String> fileNames = new LinkedList<>();
    public static Queue<String> newNameList = new LinkedList<>();
```

```
@Override
public void start(Stage primaryStage) throws Exception
{

    //define, load scenes
    Parent root = FXMLLoader.load(getClass().getResource( name: "menuScene.fxml"));
    mainMenuScene = new Scene(root,  width: 837,  height: 549);

    Parent replaceRoot = FXMLLoader.load(getClass().getResource( name: "replaceWindowScene.fxml"));
    replaceScene = new Scene(replaceRoot,  width: 600,  height: 400);

    Parent deleteRoot = FXMLLoader.load(getClass().getResource( name: "deleteWindowScene.fxml"));
    deleteScene = new Scene(deleteRoot,  width: 600,  height: 400);

    Parent insertRoot = FXMLLoader.load(getClass().getResource( name: "insertWindowScene.fxml"));
    insertScene = new Scene(insertRoot,  width: 600,  height: 400);

    Parent tagRoot = FXMLLoader.load(getClass().getResource( name: "tagWindowScene.fxml"));
    tagScene = new Scene(tagRoot,  width: 600,  height: 400);

    //build the main stage
    primaryStage.setTitle("File Renamer (Music)");
    primaryStage.setScene(mainMenuScene);
    primaryStage.show();

    //let primary stage become accessible to other classes
    openPrimaryStage =  primaryStage;

}
```

Being an Javafx application, the program should override the function “start” as the entry point. I link the GUI layout files with my java program and build the main window for the program.

```
public void refreshListView()
{
    /**Procedure
     * clear the old list of names
     * get new name list from fileInstances
     * refresh it on ListView
     *
     * refresh the new Preview
     *
     * */

    //add the names into fileNames queue
    Main.fileNames = new LinkedList<>();//first clear the original list of names
    for(File target : Main.fileInstances)// then import names from the fileInstances
    {
        Main.fileNames.offer(target.getName());
    }

    //before we add items into the original Name ListView, we have to clear it up
    while (originalNameListView.getItems().size() > 0)
        originalNameListView.getItems().remove( index: 0);//so always delete the first element, until the list size == 0


    originalNameListView.getItems().addAll(Main.fileNames);

    //refresh the new preview
    refreshPreviewList();
}
```

refreshListView function will be executed while the original file list is changed, like the moment when the user adds files into the program. The reason why I couldn’t simply use an equal sign to import all of the values from Main.fileNames to originalNameListView, the model of the original file list on GUI interface, is that the object that originalNameListView used to point will be

deleted and will pointing at the object of Main.fileNames instead. If I simply use the equal sign to transfer the values, the link between oringalNameListView and the GUI interface will be lost.

```
187     //Export Naming Setting (ENS) Button On Action
188     public void ENSButtonOnAction(ActionEvent actionEvent)
189     {
190         /** Procedure
191          *
192          *  - Open Directory selector window
193          *  - create a new .rp file and print the TTP text in it
194          *
195          * */
196
197         //save a file logic
198         FileChooser fileChooser = new FileChooser();
199         fileChooser.setInitialFileName("rename preference");
200         //set the type of file user can save
201         FileChooser.ExtensionFilter extFilter =
202                 new FileChooser.ExtensionFilter( description: "Rename Preference File (*.rp)", ...extensions: "*.rp");
203         fileChooser.getExtensionFilters().add(extFilter);
204
205         fileChooser.setTitle("Save your file");
206
207
208         //open the file chooser window, this instance will return the results
209         File selectedFile = fileChooser.showSaveDialog(Main.openPrimaryStage);
210
211         //user may click cancel, so selectedFile could be null, which will throw IOException
212         try {
213             //print the TTP into file
214             PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(selectedFile)));
215             pw.print(TTPTextBox.getText());
216             pw.close();//save the file
217         } catch (IOException e)
218         {
219             e.printStackTrace();
220         }
221
222     }
```

ENS Button on Action function will be executed while the user wants to export the naming settings in to file and push the button on the GUI. The program will provide a file selector window, and by getting a target location in their computer, the program may fetch the commands from TTP Text Box (Task that will perform text box) and export them through PrintWriter.

```
224     //Import Naming Setting (INS) Button On Action
225     public void INSButtonOnAction(ActionEvent actionEvent) throws IOException
226     {
227         /**Procesure
228          *
229          * - open directory selector and select a *.rp file
230          * - read the file and put the text into TTP TextBox
231          *
232          * */
233
234         //choose a file logic
235         FileChooser fileChooser = new FileChooser();
236         FileChooser.ExtensionFilter extFilter =
237                 new FileChooser.ExtensionFilter( description: "Rename Preference File (*.rp)",  ...extensions: "*.rp");
238         fileChooser.getExtensionFilters().add(extFilter);
239
240         fileChooser.setTitle("Choose a rename preference (*.rp) file");
241
242
243         //open the file chooser window, this instance will return the results
244         File selectedFile = fileChooser.showOpenDialog(Main.openPrimaryStage);
245         if(selectedFile == null) return; //if the user didn't select a folder, then do nothing and exit.
246
247         BufferedReader br = new BufferedReader(new FileReader(selectedFile));
248
249         String commandFromFile = "";
250         String readLine = br.readLine();
251
252
253         while(readLine != null)//as long as there are lines to read
254         {
255             commandFromFile = commandFromFile + readLine + "\n";//add the line into command
256             readLine = br.readLine();//read the next line. it could be empty.
257         }
258
259         if(commandFromFile.length() != 0)//as long as the file is not empty, import the naming setting
260             TTPTextBox.setText(commandFromFile);
261
262     }
263
```

This function will provide a gateway for user to import and reuse the naming setting they have saved on their computer. It will open a file chooser window and uses buffer reader to read the *.rp (rename preference) file and load the naming settings into TTP TextBox.

```
52      //new name list record is recorded inside a single refresh preview action
53      //whenever the new refresh preview action starts, it should be initialize with fileNames
54      Main.newNameList = new LinkedList<>();//initialize the new Name List
55      //because we don't want to see..
56      //for example, "replace 0 to 000" in "0" into "000000000..." while we press refresh several times
57      Main.newNameList.addAll(Main.fileNames);
58
59      if(mainMenuController.Pub_TTPTextBox.getText().equals(""))
60      {
61          //if the command list is empty, then the preview list should show fileNameList while the user clicks refresh
62          return;//if nothing in the text box, end the process directly
63      }
64
65      String command = mainMenuController.Pub_TTPTextBox.getText();
66
```

```
66
67          //split the commands into single command. For example, split replace(...); delete(...);
68          String[] tasks = command.replaceAll( regex: "\n",  replacement: "").split( regex: ";");
69
70          //target means single command like replace (..), now we need to identify the type of the command
71          for(String target : tasks)
72          {
73              //break down single command like replace (..) into [replace] [(...)]
74              String[] singleCommandBreakDown = target.trim().split( regex: " ");
75              //room number 0 should be the type of the command, for example, replace.
76
77              switch (singleCommandBreakDown[0].toLowerCase(Locale.ROOT))
78              {
79                  case "replace":
80                      replace(target);//finished
81                      break;
82                  case "delete":
83                      delete(target);//finished
84                      break;
85                  case "insert":
86                      insert(target);
87                      break;
88
89                  default:
90                      System.out.println("No Valid Command Detected");
91              }
92          }
93          //update the ListView on GUI
94      }
```

The function “processSettingText” is one of the most important function in this program, because it will load the naming setting from TTP TextBox and call the functions to dispose those commands respectively. The function includes a detection of empty TTP TextBox and, as this function will be executed whenever the user click “refresh preview” button, but still have to support multiple lines of commands, the source of file name must be identified and initialized carefully.

```
3   import org.jaudiotagger.audio.AudioFileIO;
4   import org.jaudiotagger.audio.exceptions.CannotReadException;
5   import org.jaudiotagger.audio.exceptions.InvalidAudioFrameException;
6   import org.jaudiotagger.audio.exceptions.ReadOnlyFileException;
7   import org.jaudiotagger.tag.FieldKey;
8   import org.jaudiotagger.tag.TagException;
9
10  import java.io.File;
11  import java.io.IOException;
12  import java.util.LinkedList;
13  import java.util.Locale;
14  import java.util.Queue;
15
```

The function “insert”, as implementing the feature to read tags of music files, the use of external library “jaudiotagger” is a necessity.

```
267         //if the command contains the use of tags expression, we should indicates that
268         if(theText.contains(TagIndicatorText))
269         {
270             System.out.println("USED TAG");
271             useTag = true;
272             //delete the tag Indicator [\\$`renamer`!#useOfTag#\\$]
273             theText = theText.substring(TagIndicatorText.length());
274
275             if(theText.equals(supportedTagType[0])) tagType = FieldKey.TRACK;//TRACK Info
276             else if (theText.equals(supportedTagType[1])) tagType = FieldKey.ALBUM;
277             else if (theText.equals(supportedTagType[2])) tagType = FieldKey.ARTIST;
278             else System.out.println("\n\n!!!!!!!!!!!!!!!! You've declared the use of Tag, " +
279                     "but the type \"" + theText + "\" is unsupported\n\n");
280         }else System.out.println("NO USE OF TAG");
```

However, insert function may or may not contain the use of tag, but it will be indicated in the command, so the program should mark it if the command requires the use of tag. The routine above will mark the command that contains a tag indicator and save the type of tag.

```
288         //preload the data
289         Queue<String> originalNameList = Main.newNameList;//load original name list
290         Main.newNameList = new LinkedList<>();// clear the new name list to put new names
291
292         Queue<File> fileInstancesCOPY = new LinkedList();
293         fileInstancesCOPY.addAll(Main.fileInstances);
294
295         //insert position may vary because I implemented a feature to trim the insertPosition if the position index is
296         // larger than the length of the file Name,
297         // so I decided to let the insert position vary according to the maximum length of the file name
298         // ,therefore we need to save the original number of the insert position
299         // so we can change the insert position as we want along different file names.
300         int originalInsertPosition = insertPosition;
301
302         for(int indx = 0; indx < Main.fileInstances.size(); indx ++)
303         {
304             String originalName = originalNameList.poll();
305
306             insertPosition = originalInsertPosition;//initialize the insert Position
307
308             if(insertPosition > originalName.length())//if the position index is longer than the file name,
309                 insertPosition = originalName.length(); //we shall trim the insert position
310
311             if(useTag)
312             {
313                 try {
314                     theText = (AudioFileIO.read(fileInstancesCOPY.poll())).getTag().getFirst(tagType);
315                     if(theText.length() == 1) theText = "0" + theText;//and 0 before 1-9, so it became 01-09
316                 }catch (IOException e) {
317                     e.printStackTrace();
318                 } catch (CannotReadException e) {...} catch (ReadOnlyFileException e) {
321                     e.printStackTrace();
322                 } catch (TagException e) {
323                     e.printStackTrace();
324                 } catch (InvalidAudioFrameException e) {
325                     e.printStackTrace();
326                 }
327             }
328             Main.newNameList.offer( e: originalName.substring(0, insertPosition) + theText
329                     + originalName.substring(insertPosition, originalName.length()));
330         }
```

The picture above presents the part of insert function that process the new file name after applying naming rules by inserting correct text to correct position. As there are lots of potential error could happen while fetching the tag information, the code is surrounding by try/catch.

```
337     public static void renameTheFile()
338     {
339         for(File target : Main.fileInstances)
340         {
341             target.renameTo(new File( pathname: target.getParent() + "\\" + Main.newNameList.poll()));
342         }
343         System.out.println("Rename Finished");
344
345     }
346
```

The renameTheFile function will be executed as the user clicks the “RTFButton”. The function uses “renameTo” function to rename the file respectively.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<GridPane alignment="center" hgap="10" vgap="10" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:cont
    <columnConstraints>
        <ColumnConstraints />
        <ColumnConstraints />
    </columnConstraints>
    <rowConstraints>
        <RowConstraints />
    </rowConstraints>
    <children>
        <AnchorPane prefHeight="549.0" prefWidth="837.0">
            <children>
                <Button fx:id="replaceButton" layoutX="55.0" layoutY="332.0" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-
                <Button fx:id="deleteButton" layoutX="55.0" layoutY="433.0" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-I
                <Button fx:id="readMusicButton" layoutX="197.0" layoutY="332.0" maxHeight="-Infinity" maxWidth="-Infinity" minHeight
                <Button fx:id="insertButton" layoutX="197.0" layoutY="433.0" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-
                <Label layoutX="55.0" layoutY="36.0" prefHeight="17.0" prefWidth="196.0" text="Task That will Perform (TTP)" />
                <ListView fx:id="previewNameListView" layoutX="570.0" layoutY="115.0" prefHeight="348.0" prefWidth="217.0" />
            <ListView fx:id="originalNameListView" layoutX="346.0" layoutY="115.0" prefHeight="348.0" prefWidth="217.0" />
            <Button fx:id="INSButton" layoutX="55.0" layoutY="257.0" mnemonicParsing="false" onAction="#INSButtonOnAction" prefHeigh
            <Button fx:id="ENSButton" layoutX="197.0" layoutY="257.0" mnemonicParsing="false" onAction="#ENSButtonOnAction" prefHeig
            <Button fx:id="RTFButton" layoutX="436.0" layoutY="481.0" mnemonicParsing="false" onAction="#RTFButtonOnAction" prefHeig
            <Button fx:id="openFolderButton" layoutX="346.0" layoutY="53.0" mnemonicParsing="false" onAction="#openFolderOnAction" p
            <Button fx:id="openFileButton" layoutX="455.0" layoutY="53.0" mnemonicParsing="false" onAction="#openFileOnAction" prefH
            <Button fx:id="refreshPreviewButton" layoutX="563.0" layoutY="53.0" mnemonicParsing="false" onAction="#refreshPreviewOnA
            <TextArea fx:id="TTPTextBox" layoutX="55.0" layoutY="58.0" prefHeight="190.0" prefWidth="247.0" />
            <Button fx:id="clearListButton" layoutX="675.0" layoutY="53.0" mnemonicParsing="false" onAction="#clearListButtonOnActio
            </children>
        </AnchorPane>
    </children>
</GridPane>
```

The picture above shows a part of the layout file of GUI in the main Menu.

Word count: 658

API used:

Jaudiotagger

Jthink.net. 2021. *JThink*. [online] Available at: <http://www.jthink.net/jaudiotagger/> [Accessed 13 January 2021].