Final Project

Object-Orientated Programming

Project Name: “Muse”

Student Name: Peter Nelson Subrata

Student ID: 2502023562

Class: L2CC

Table of Contents

A. Description

I. Introduction

When deciding for a project for my OOP class, I tried thinking about all of the things that were possible using Java. I had thought about making many things such as another game like what I made last year with ChessPy. In the end though, I chose to make a Java mp3 player as it allows me to view a different kind of project and it also correlates with my love of music.

I began this project on the 23rd of May 2022. I chose to use the Eclipse IDE for this project and I will be posting all of the code along with this report on my repository which can be accessed through my GitHub account here:

https://github.com/PewterZz/Muse---A-Java-Music-Player

II. The function of this program

The function that this program serves is that of a music player that can play any viable mp3 files from a dedicated music folder. It can play music, pause music, start over the music and go the next or previous track in the list.

The GUI provides sufficient details on what each button and action does, and all other information regarding the mp3 file is shown on the middle of the GUI. It shows things like, the album, the artist, the cover art (if present) and the name of the song. There is also an option to speed up the track up to 1000% or even vice versa by slowing it down to 25%.

B. Design

I. Parts of the program and requirements

Requirements:

- JavaFX
  - Used to design the overall application and visualize it into a window.
  - Can be downloaded from https://openjfx.io/openjfx-docs/

- SceneBuilder
  - Used to design the window of the program as it can view and edit .fxml files.
  - Not Completely necessary to run the program, but I used this to make designing the program far easier.

In this program I used a window to display all of the functions of the program. The window starts out with the starting screen which shows you the main functions such as play, pause, start over, next, and previous. Once clicked the program will execute the function that was displayed. The executed function can change the display whatever is needed to display.

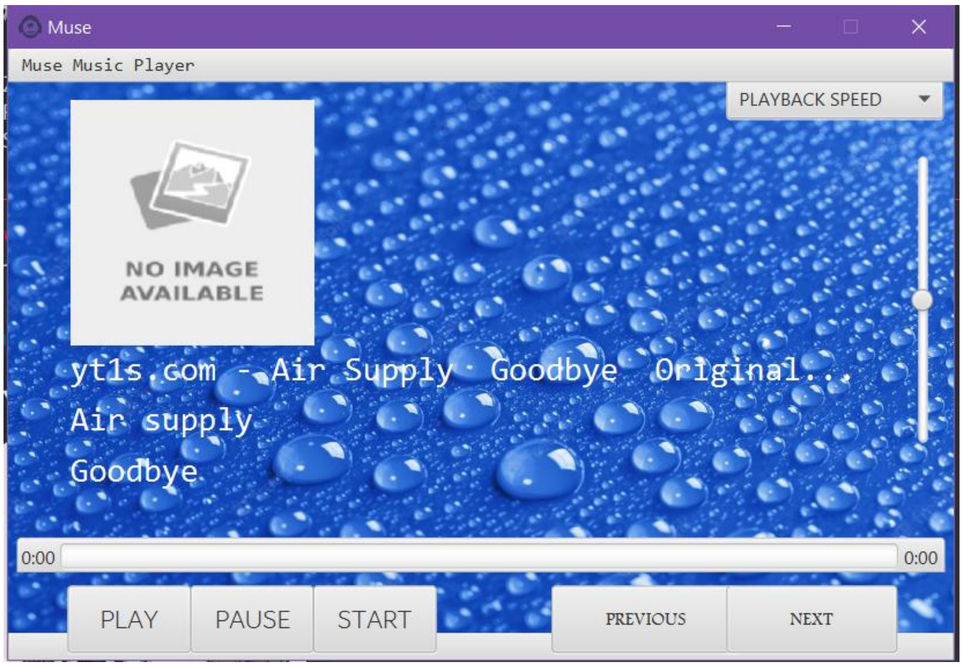
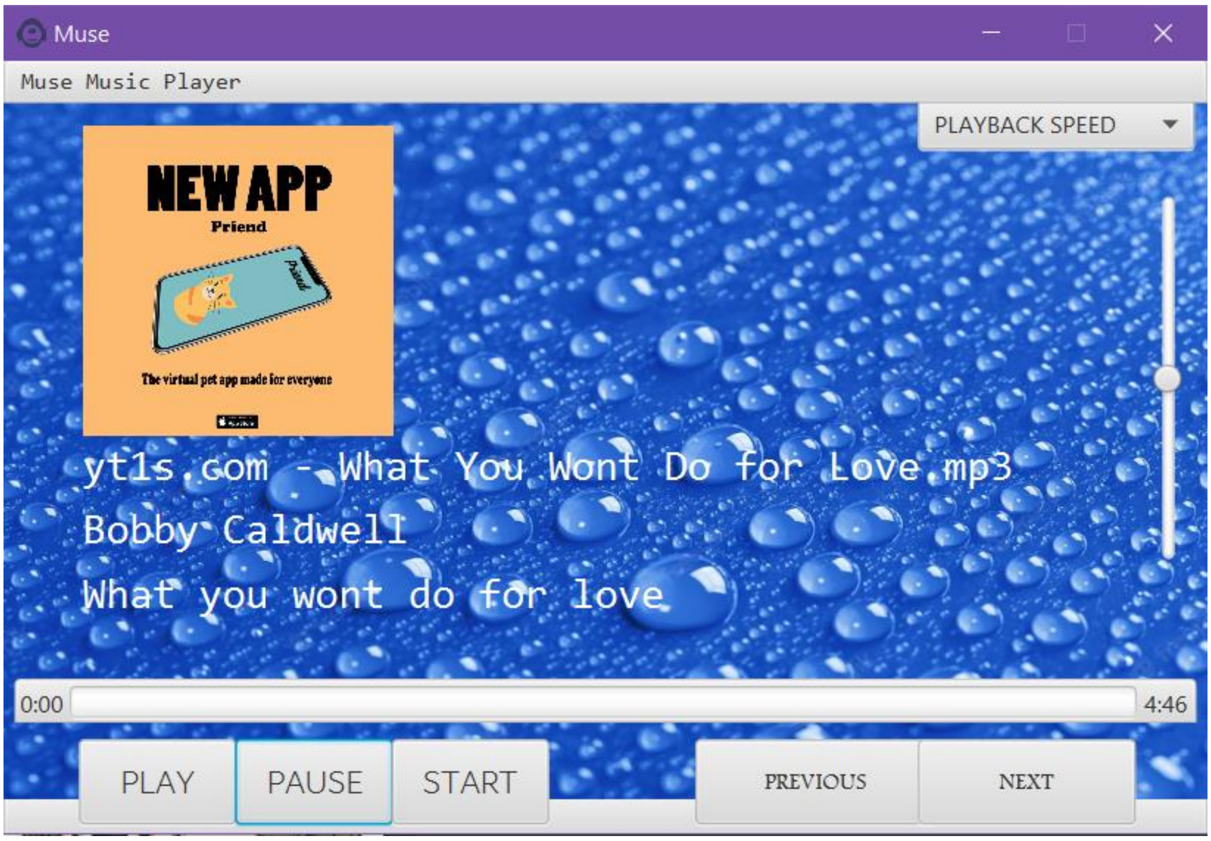


Figure 1: Starting screen for window

II. Function of each part of the program

Music must be placed in the music folder in the form of mp3 files. The program will find those mp3 files and attempt to play them on the running program. There are buttons on every side of the screen that have helpful identifiers to make sure the user knows what they do. There is a volume slider on the right side of the window that can adjust the volume of music. The program will scan the metadata of any mp3 file you put in, and if it detects that there exists some of these specific labels, it will seek to change the text according to the metadata as seen below on Fig..



yt1s.com - What You Wont Do for Love.mp3
Bobby Caldwell
What you wont do for love

Figure 2: Window showing the program being run and the buttons.

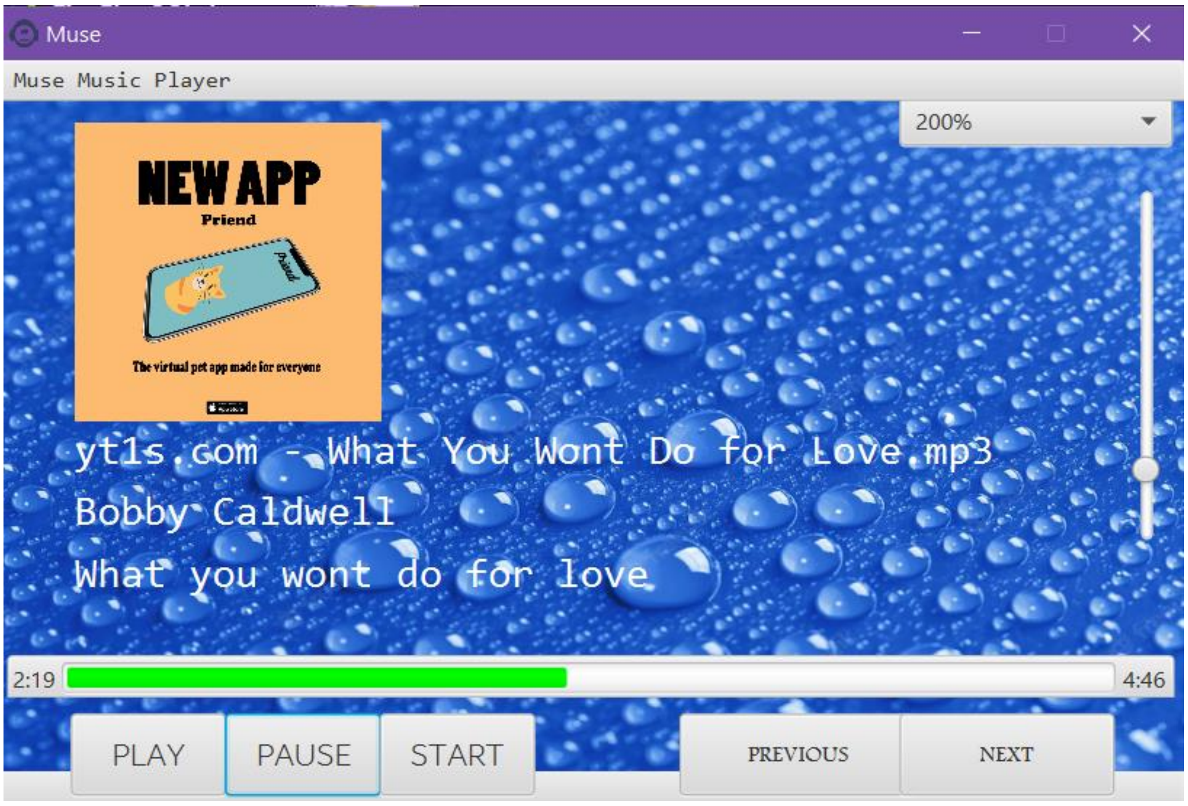

Muse
Muse Music Player
200%
NEW APP
Priend
The virtual pet app made for everyone
yt1s.com - What You Wont Do for Love.mp3
Bobby Caldwell
What you wont do for love
2:19
4:46
PLAY
PAUSE
START
PREVIOUS
NEXT

Figure 3: Window showing metadata reading and displaying.

C. Implementation

I. Classes Diagram

Main
+start(Stage stage)

Figure 4: UML Class diagram of the Main class.

Controller <interface>
-pane: Pane -songLabel : Label -songLabel1 : Label -songLabel2 : Label -endLabel : Label -startLabel : Label -playButton : Button -pauseButton : Button -startButton : Button -previousButton : Button -nextButton : Button -speedbox : Button -volumeSlider : Button -songProgressBar : Button -songImage : ImageView -songImage1 : ImageView -media : Media -mediaPlayer : mediaPlayer -directory : File -files : File -songs : ArrayList<File> songs -songNumber : Int -speeds : int[] -Timer : timer -TimerTask : task -running : boolean

-handleMetadata(String, Object)
+initialize()
+setTime()
+playMedia()
+pauseMedia()
+startMedia()

+previousMedia()
+nextMedia()
+changespeed(ActionEvent)
+beginTimer()
+cancelTimer()

Figure 5: UML Class diagram of the chesspieces class.

II. Flowcharts

Main class flowchart

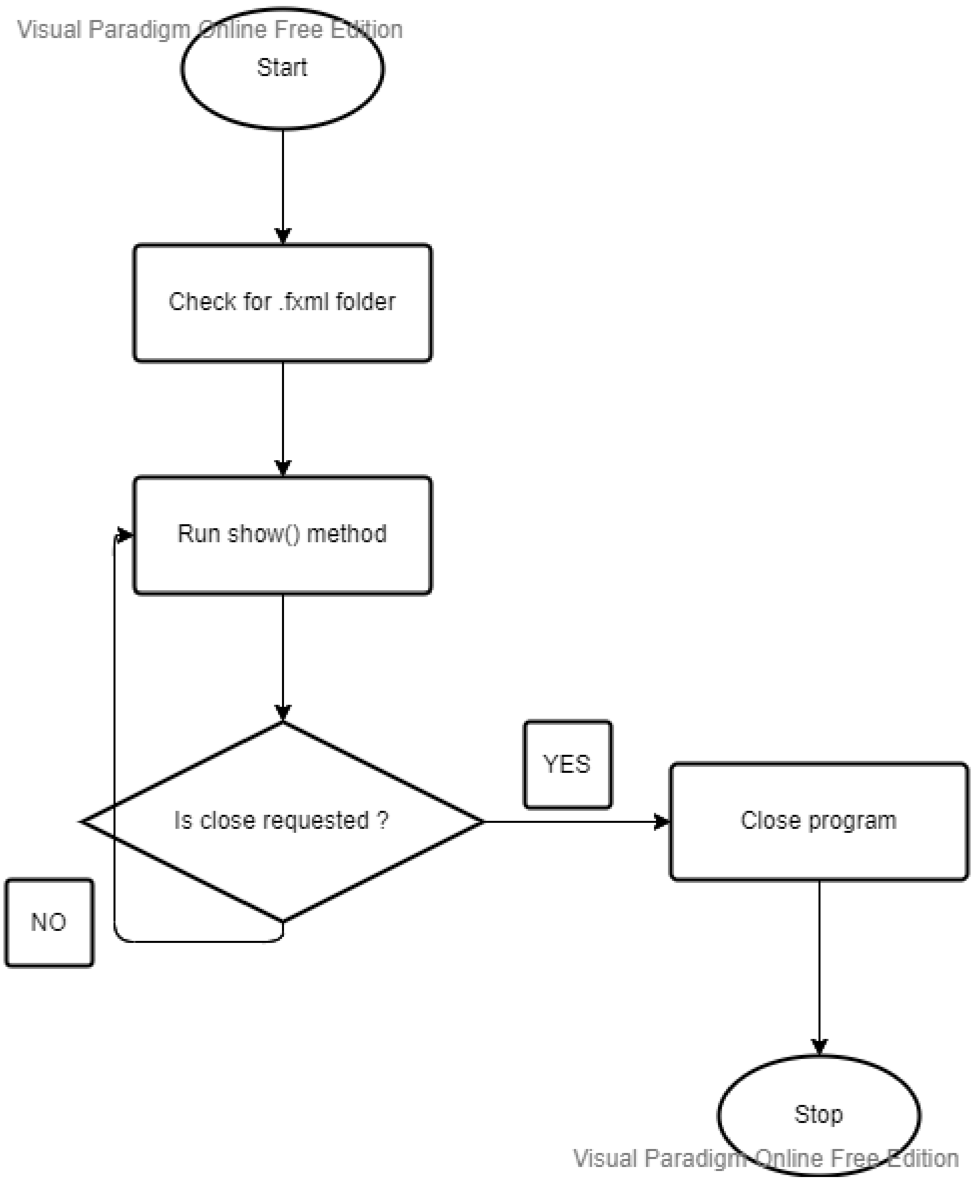


Figure 7: Flowchart of starting screen before main process.

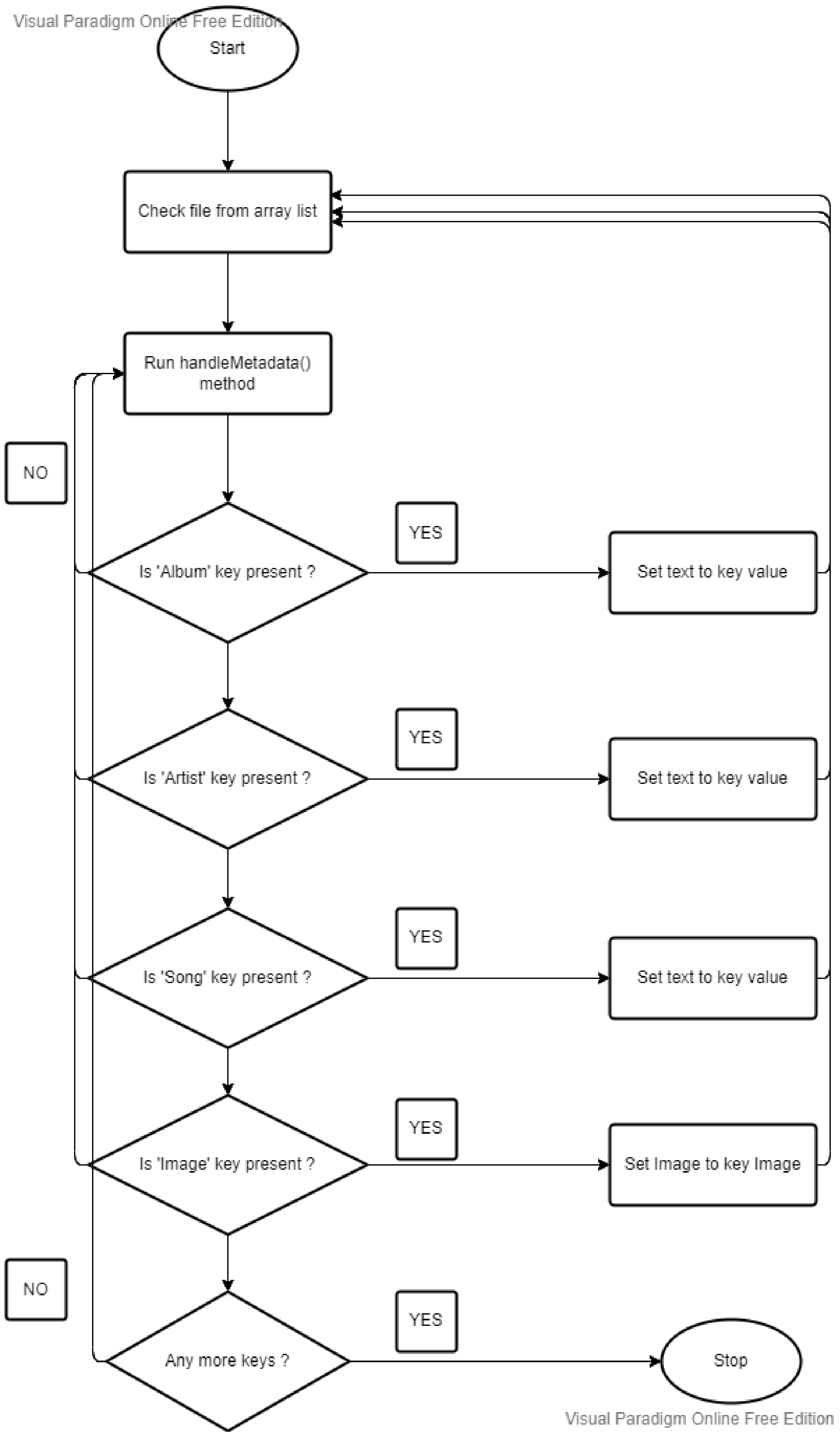


Figure 8: Checking for metadata in the file.

III. Extensibility

1. Comment lines are used to to further explain the functions of each function and method and the purpose of each of them to help other programmers.
2. Variables use meaningful identifiers to help other coders to understand easier.
3. Class diagram and flowchart to make it easier for others to understand and to easily identify features.

IV. Explanation of all the functions made and used

Main.java

- start(Stage stage):
  - when application is launched this method creates a stage, scene and root to be shown.
  - The stage is the first layer of the window and the scene comes afterwards followed by the root which includes all of our fxml objects.
- main(String[] args):
  - Main is called to start the application.

Controller.java

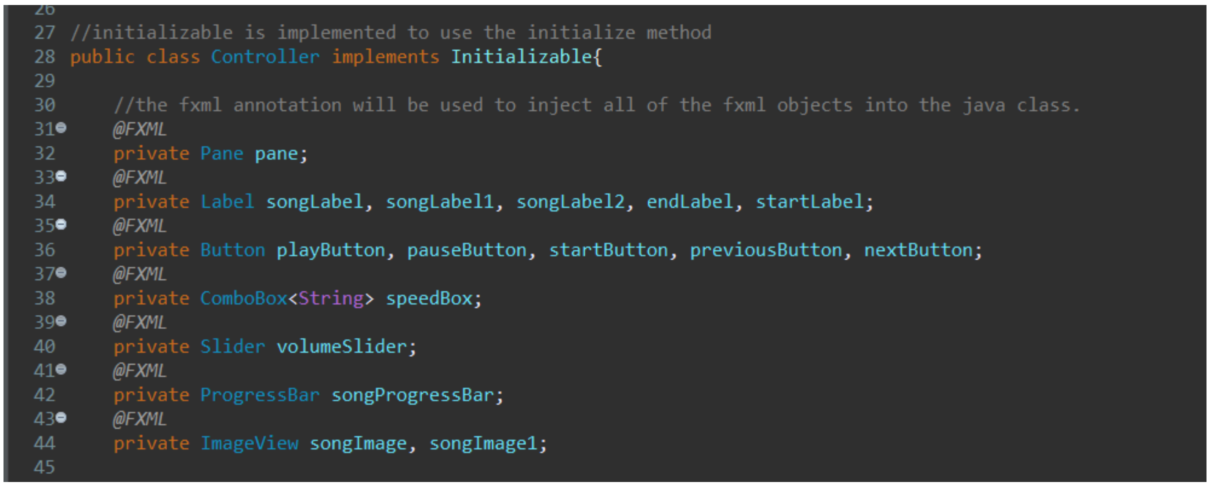
- handleMetadata(String key, Object value):
  - Handle metadata takes a string and object corresponding to the key and value of the map that is returned when getMetadata is called on a media object.
  - Metadata information is set when the keys are found and the object is converted to a string to display it on the labels.
- initialize(URL arg0, ResourceBundle arg1):
  - Serves as the main method of the controller class that will run upon the application being launched.
  - Adds the values of speed adjusting into the combo box.
  - Array list is made to store the file paths of the music files into it.
- setTime():
  - Sets the current time and duration of the song onto the labels provided.
- playMedia():
  - Plays the current media mp3 object and starts the timer for the progress bar.
- pauseMedia():
  - Stops playing the current media mp3 object.

- startMedia():
  - Regresses the song back to the start of the duration.
- previousMedia():
  - Checks array list for the previous element in it, then proceeds to play the song. As well as set the metadata appropriately.
- nextMedia():
  - Checks array list for the next element in it, then process to play the song as well as set the metadata appropriately.
- changeSpeed():
  - Changes the rate at which the song is being played at.
- beginTimer():
  - Starts a timer that will run throughout playing the song that will also include an anonymous function to change the progress bar depending on the value of the current time divided by the total duration.
- cancelTimer():
  - Stops the timer and pauses the progress bar.

D. Lessons learned

I. Using Pygame

JavaFX is the software I used to make this game and as it was not a part of the main topic of the class, I had to learn it by watching tutorials and reading the documentation all of which was provided on the internet. Using JavaFx at first was very frustrating and annoying, even downloading JavaFx was quite the challenge as Java started to act weirdly sometimes. After having used JavaFx for a while now though, I have seen that it can be really good for making an application window and when making the mp3 player because of the tutorials and help I got from people around the internet I was able to make my final project for semester 2.

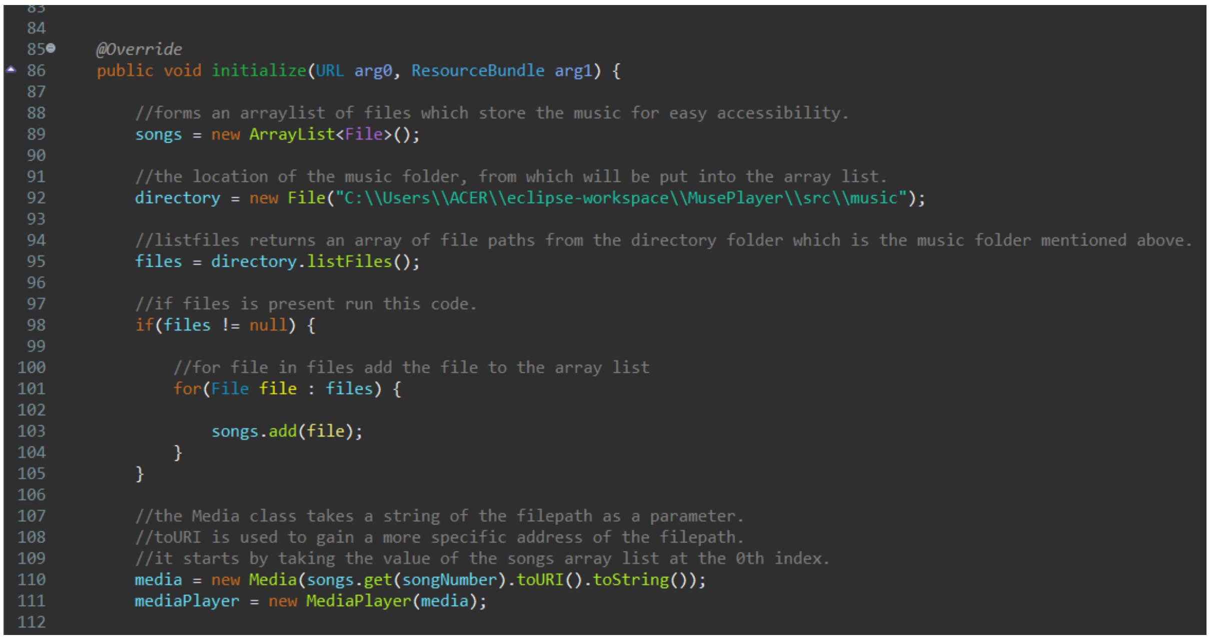


```java
//initializable is implemented to use the initialize method
public class Controller implements Initializable{

    //the fxml annotation will be used to inject all of the fxml objects into the java class.
    @FXML
    private Pane pane;
    @FXML
    private Label songLabel, songLabel1, songLabel2, endLabel, startLabel;
    @FXML
    private Button playButton, pauseButton, startButton, previousButton, nextButton;
    @FXML
    private ComboBox<String> speedBox;
    @FXML
    private Slider volumeSlider;
    @FXML
    private ProgressBar songProgressBar;
    @FXML
    private ImageView songImage, songImage1;

```

Figure 9: Snippet of code used to declare all of the FXML variables and objects.



```java

    @Override
    public void initialize(URL arg0, ResourceBundle arg1) {

        //forms an arraylist of files which store the music for easy accessibility.
        songs = new ArrayList<File>();

        //the location of the music folder, from which will be put into the array list.
        directory = new File("C:\\Users\\ACER\\eclipse-workspace\\MusePlayer\\src\\music");

        //listfiles returns an array of file paths from the directory folder which is the music folder mentioned above.
        files = directory.listFiles();

        //if files is present run this code.
        if(files != null) {

            //for file in files add the file to the array list
            for(File file : files) {

                songs.add(file);
            }
        }

        //the Media class takes a string of the filepath as a parameter.
        //toURI is used to gain a more specific address of the filepath.
        //it starts by taking the value of the songs array list at the 0th index.
        media = new Media(songs.get(songNumber).toURI().toString());
        mediaPlayer = new MediaPlayer(media);

```

Figure 10: Snippet of code to show the main process.

II. Using the SceneBuilder Extension

I used the SceneBuilder extension when designing the GUI for my program as it was recommended to me at the time when I was trying to find projects to do. Using scenebuilder proved to be very flexible and I enjoyed working with it.

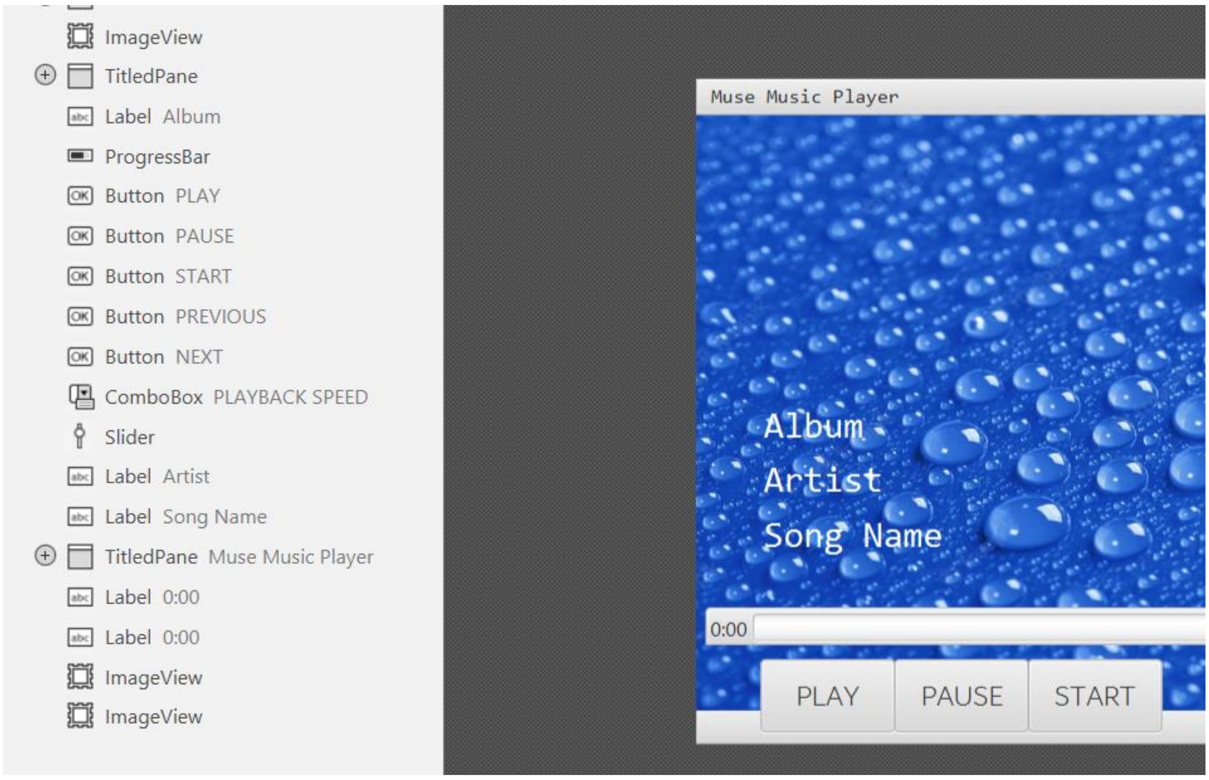


Figure 11: Snippet of code to show working with SceneBuilder.

III. Error Handling and Debugging

One of the most frustrating parts when becoming a coder is having to deal with debugging your own program. I feel the same way too, as having to solve a problem that you caused can be quite confusing. I have devised my own way to figure out what went wrong in my code to lead to something not going the way I want to. The first method I did is writing down a System.out.println function when I expect things to be one way but it turns out to be wrong. If it is printed out on the console that means that I was right in thinking so, if not then something clearly went wrong and I will need to check elsewhere to solve the problem. These functions come very handy when having to deal with something that doesn't seem to be that big of a deal. I used multiple of these functions when some sort of error occurred that doesn't seem to be because of a logic error. Finally, if all things mentioned before didn't work, I would probably try to look for a solution online and if that still didn't work then I'd have to read the documentation of any external modules I was using for a full understanding of all of the functions included.

E. Evaluation

I. Does the program work properly?

The JavaFX program is essentially just a simple mp3 music player, that currently does not really have anything different to do from your standard music player. The program works really well as a game and has basically no errors that can occur that hasn't been dealt with. The program is also really fast with determining the next song and other things as well. Overall, I would say that this program is good in terms of not being to complicated to understand its features as well its functionality.

II. Future Improvements that can be done

I had thought of making a audio visualizer but thought that it would be very hard to implement and explain, but besides that I could manipulate the UI to be better and also provide a better display to show all of the songs that are available in the music folder.

III. Reflection

I feel as if with this project I have grown stronger as a programmer and I wish that I can never stop growing my skills. With that being said I am satisfied with how my program ended up being, even though I know that there are many improvements that can still be done with the existing program. I will continue to work on this program after submitting this report and I will keep improving it so that it can be something that I am proud of. After all this is the first Java project I have done and I will remember the lessons, I learnt when making this all throughout my career.

F. Evidence of working program

I. Testing and pictures

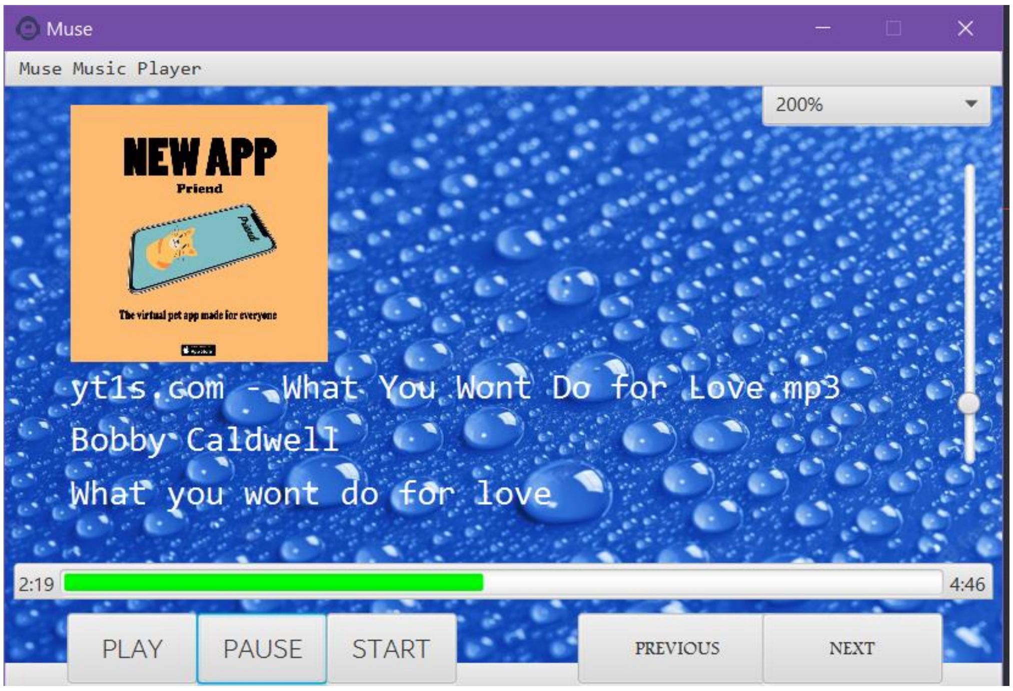


Figure 12: Starting screen of the program.

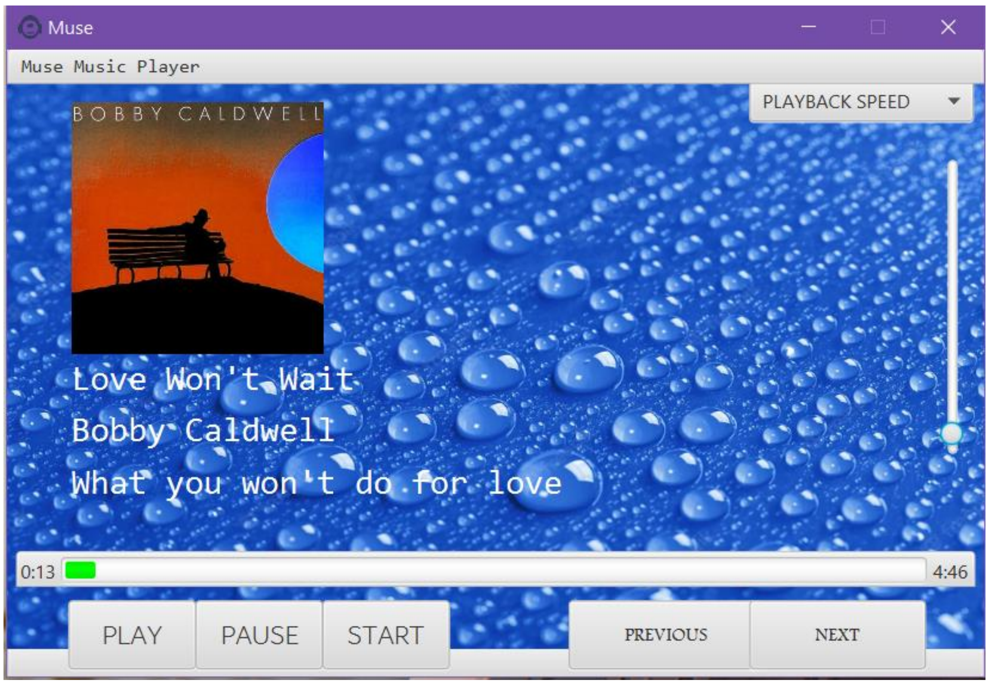


Figure 15: Main process of the game.