****Tarlac State University

**COLLEGE OF COMPUTER STUDIES**

Case Study

in

CC3

Submitted By:

**Petines, Marc Jhasper L.**

**Cervantes, Elijah Gabe C.  
Valencia, Earl Benjie R.  
Quismundo, Denzel Matthew P.**

Submitted To:

**Rovell F. Asidera**

Date:  
**May 17, 2023**

**INTRODUCTION**

Music players have revolutionized the way we experience and enjoy music, providing us with a portable and immersive listening experience.   
By definition, a music player, also known as a media player, is a device or software application designed to play audio files, allowing users to listen to music and other forms of audio content. It provides a platform for organizing, accessing, and playing various types of audio formats, including but not limited to MP3, WAV, FLAC, and AAC.

According to Techopedia, a music player is defined as "a software program or hardware device that allows users to play audio files and manage their music libraries" (Techopedia, n.d.). It goes on to explain that music players often include features such as playlist creation, playback controls (play, pause, skip), equalizer settings, and in some cases, visualizations.

Furthermore, an article by Lifewire, a trusted technology information source, expands on the definition by stating that music players can be standalone devices, like portable MP3 players, or integrated into other devices such as smartphones, tablets, and computers (Bell, 2022).

Music players have revolutionized the way we experience and enjoy music, providing us with a portable and immersive listening experience. According to a study published in the Journal of New Music Research, music players have played a significant role in shaping our musical culture and consumption habits (Cohen & Shapira, 2017). From the emergence of phonographs in the late 19th century to the rise of portable MP3 players in the 21st century, these devices have become integral to our daily lives.

In conclusion, music players have not only transformed the way we listen to music but have also shaped the entire music industry. Their evolution, from vinyl records to digital streaming platforms, has revolutionized our access to music and influenced our musical preferences. And fascinated by the said topic, what if we made our own music player using our knowledge learned in this subject? Let’s find out how we made it.

**I. The Main Frame and Label**

To start off this program, we made a blank canvas of this program by implementing a JFrame and making a simple label using JLabel on this program.

A screenshot of a music player

Description automatically generated with medium confidence

***(This is the full interface of our program where you can see the different part implemented which will be explained on the next chapters. But we highlighted on the screenshot the name of the program and the label which was mentioned.)***

**Snapshots of the codes for this chapter:**

Like any other code, it is vital to import the packages and classes needed for this project which is namely Java Sound, Swing, AWT and io.File.

A screen shot of a computer program

Description automatically generated with low confidence

And then we initialize **JComponents** which are the JFrame named aFrame and a JLabel named appProject.

A picture containing text, font, screenshot, line

Description automatically generated

Next we declare our JFrame named aFrame and set the bounds, layout and other specifics for our JFrame.

***Code:***

A screen shot of a computer code

Description automatically generated with low confidence

And then, we declare also our JLabel named appProject and set the bounds, layout, and other specifics for our JLabel.

***Code:***

A screen shot of a computer program

Description automatically generated with low confidence

**II. Sidebar Panel and Cover Panel**

The cover panel and sidebar panel play a crucial role in the design of the GUI Application. Most importantly, the sidebar panel also holds the *JComponent* JList Playlist, which will be further discussed in Chapter IV. In the screenshots below, shows the aesthetic design of the sidebar panel and cover panel.

***(The cover panel UI has an aesthetically pleasing gradient purple background and the application name is displayed in this panel.)***

A purple and white background with white text

Description automatically generated with low confidence

A screenshot of a music album

Description automatically generated with low confidence

***(The sidebar panel UI displays the “ALBUM” label, along with the album cover and album name.)***

**Snapshots of the codes for this chapter:**

Before setting up the *bounds, background and foreground color, text* etc. of the JPanels and JLabels. It is vital to declare or initialize JComponents.

A screenshot of a computer program

Description automatically generated with medium confidence

For additional information, the initialization of these components is outside of the main method and set the access specifier to public, leading to the conclusion that the other user defined methods can access the *JComponents*. As seen in the screenshots below, the JComponents are declared as static because the main method cannot access the non-static *JComponents*.

In this GUI application, the Sidebar Panel is divided into four sub panels:

***FirstSP*:** First Subpanel

***SecondSP*:** Second Subpanel

***ThirdSP*:** Third Subpanel

***FourthSP*:** Fourth Subpanel

*(Take note: the* ***firstP*** *is different from the* ***firstSP****.)*

**FirstP**

This component contains the *JLabel album* and *firstP* represents and displays the first *MAIN* panel of the sidebar. The foreground color for the label is white and the background color for this panel is grayish black.

***Code:***

**A screen shot of a computer program

Description automatically generated with medium confidence**

**FirstSP, SecondSP and ThirdSP**

The vertical (firstSP and secondSP) and horizontal (thirdSP) bar components that encloses the *CoverLabel*. The background color of this components are the same from the *firstP JPanel*.

***Code:***

**A screen shot of a computer program

Description automatically generated with medium confidence**

**A screen shot of a computer program

Description automatically generated with low confidence**

**A screen shot of a computer program

Description automatically generated with low confidence**

**FourthSP**

Even though the background color is the same from other *JPanels*. This JPanel holds a special *JComponent* called *JList Playlist* (Chapter IV) which is responsible for displaying .wav music files (Array List).

***Code:***

A screen shot of a computer program

Description automatically generated with low confidence

**albumCoverLabel**

A specialized *JLabel component* that contains a jpg file. There is also another *JLabel* included that is called *nPlaylist* because it pertains to the name of the certain Playlist. In this particular part of the source code shows the usage of *setIcon()* method.

*Tip: If you want a certain text to appear in an Image, make sure that the setBounds() of the label is ALIGNED to the other label.*

A screen shot of a computer program

Description automatically generated with low confidence***Code:***

**CoverLabel1 *(Formerly known as the Second Main Panel)***

In the early ***prototype*** designs, ***CoverLabel1*** is a JPanel called ***secondP***. However, the group decided to change the Panel to a ***background image*** for aesthetic or design purposes. The **JLabel CoverLabel1** has the same application of ***setIcon( )*** method. There is another ***JLabel*** called ***ApplicationName*** which is added to the ***CoverLabel1***.

***Code:***

**CoverLabel1 *(Formerly known as the Second Main Panel)***

In the early ***prototype*** designs, ***CoverLabel1*** is a JPanel called ***secondP***. However, the group decided to change the Panel to a ***background image*** for aesthetic or design purposes. The **JLabel CoverLabel1** has the same application of ***setIcon( )*** method. There is another ***JLabel*** called ***ApplicationName*** which is added to the ***CoverLabel1***.

***Code:***

A screen shot of a computer code

Description automatically generated with low confidence

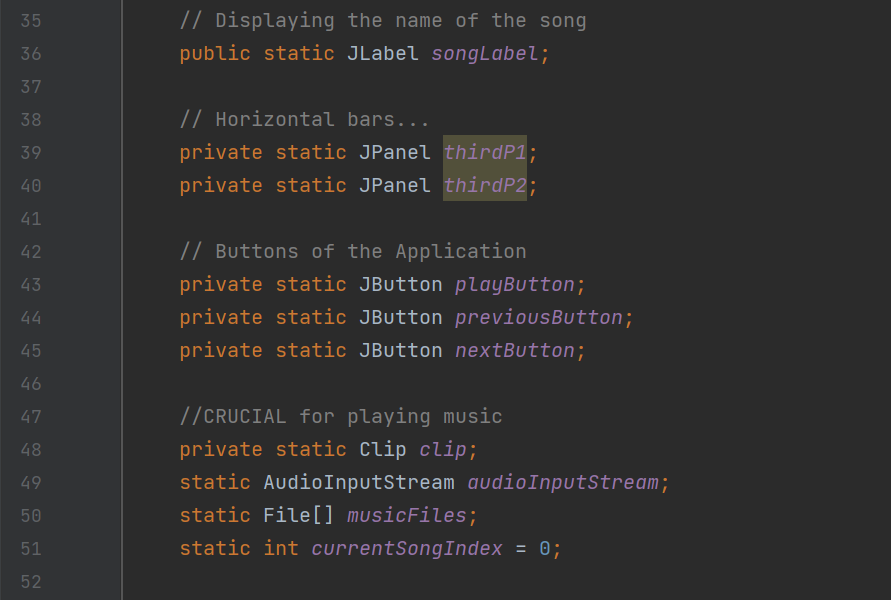
**Conclusion:**

*These are the JComponents that are declared in the Sidebar and the CoverPanel also known as CoverLabel of the GUI application.*

**III. Buttons**

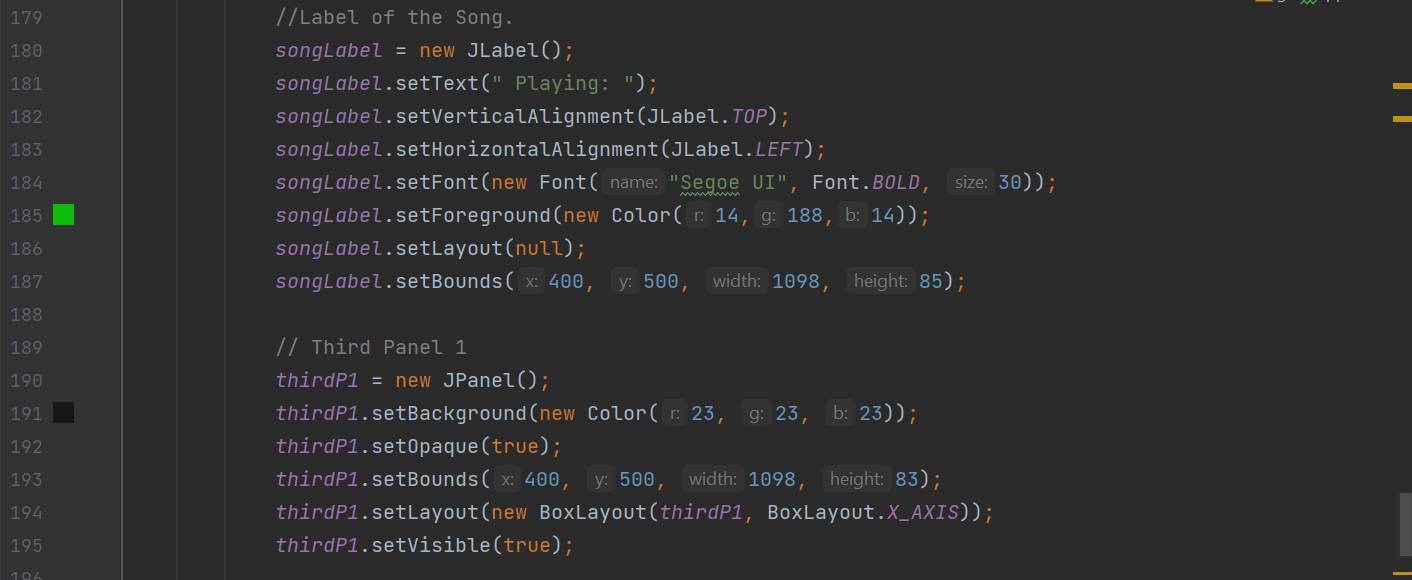
**Snapshots of the codes for this chapter:**

So of course, first of all we will declare our variables that we are going to use, specifically the: *JPanels, JButtons, JLabels, Clip, AudioInputStream, File[], and the CurrentSongIndex.*



Now the panels will serve as the space where we will put the buttons, now the buttons do is; play, pause, previous, and next, but in this case the pause and play buttons are in the same button.

Now here is where we will manipulate the panel to be whatever we want it to be. And of course, this will be at the bottom of our GUI, with set boundaries so that it will not overlap with our other panels.



Next is we set our label in the top left corner of our panel with the label **“Playing: “**now why this you ask? It’s because the label is not visible until a song is playing.

Now for our second third panel, why 2 third panels? Well, here they are on top of each other, but they look like one panel, why use 2? It’s so that the other one contains the label and the other one contains the buttons.

***:***

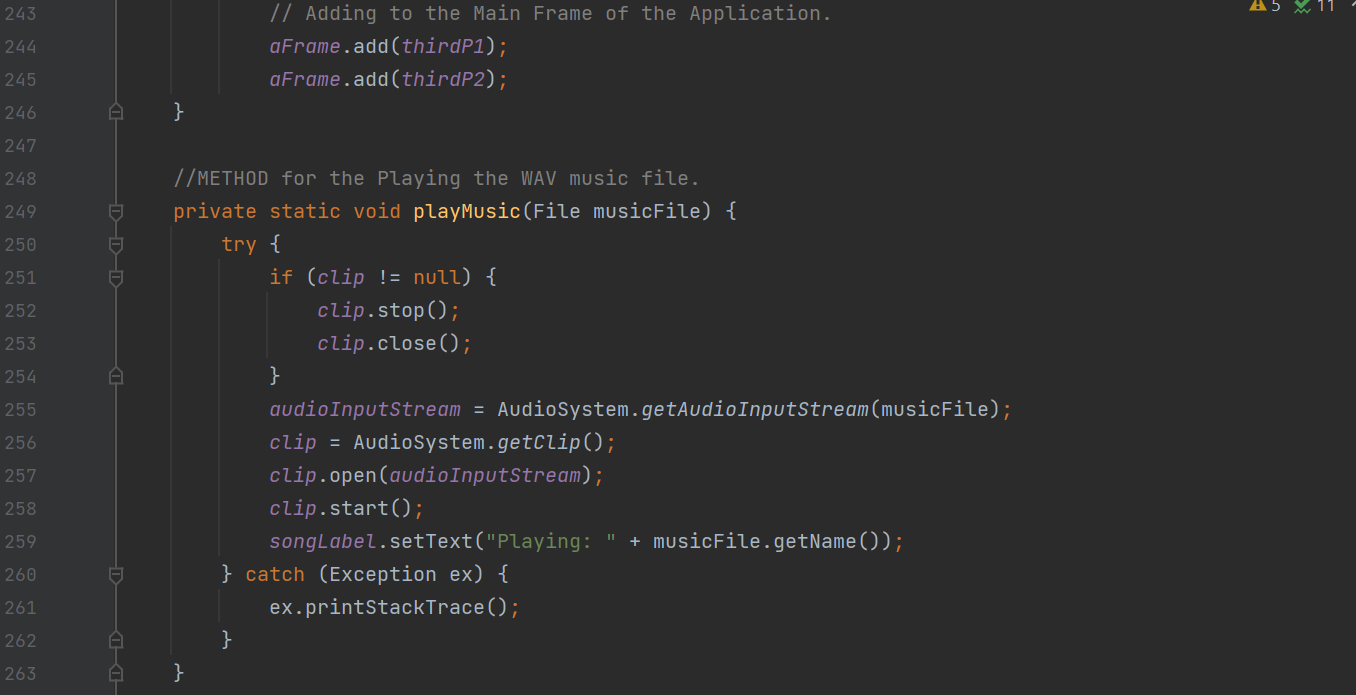
Now here you will see where our file path is declared here we used a specific file path so that it won’t get the 2 folders of the same name, why you ask? Because our GUI only plays **WAV** music files only.

We can also see here our buttons being created with the “play/pause” as their label, with an addition of giving the button the action listener command.

Now here we see our next and previous buttons getting created with the label **“Previous”** and **“Next”** respectively. At the bottom we can see the buttons getting added to our second third panel.



***Code:***



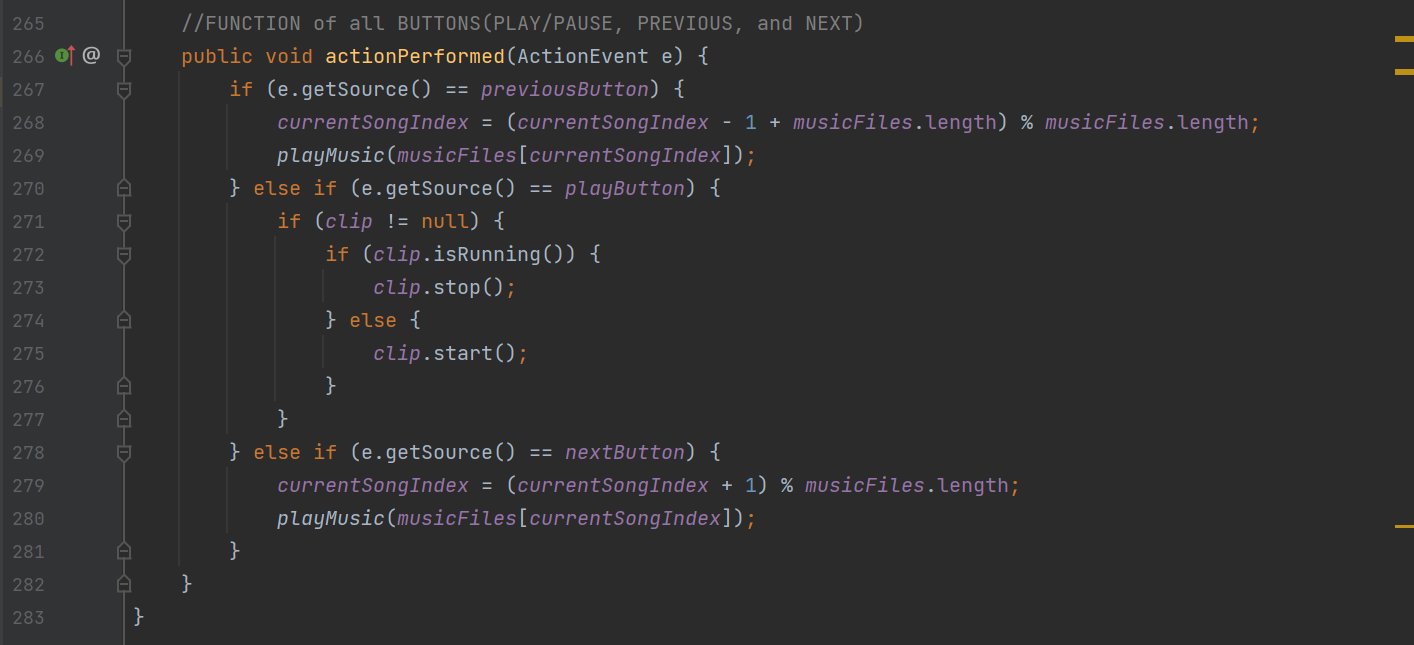
Here at the top, we can see the third panel getting added to our frame.

Then here at the button we do a **try and catch** statement where the program will test the try block of code for any errors and our catch statement displays what the error is and from what line.

Our try block of code where if our clip or song that is plying is not equal to nothing we will stop the current clip and close it just in case to not overlap 2 songs together, and after that it will get the music file, get it’s audio and will start the music, and this is where the Label earlier will be visible and display what song are we playing.

Now in this section is where we handle the actions our **Next, play/pause and Previous button.**

***Code:***



Here we can see that we use the **actionPerformed** class for our actions where the **ActionEvent** is our variable e.

Now if the button that was clicked was the **Previous** button the code simply scans the **currentSongIndex** and minus 1 plus music file length in accordance with how many songs are left in the folder, and simply plays the previous song. If the **play/pause** button is pressed it will simply do the if/else statement, we have here and scan if there is a song playing and stop or start it if nothing is playing.

Now if the button that was clicked was the **Next** button the code simply scans the currentSongIndex and plus 1 plus music file length in accordance with how many songs are left in the folder, and simply plays the next song.

**IV. The Playlist**

This is the songbook of our music player where we pick our desired song to be played for our music player.

A screenshot of a video game

Description automatically generated with low confidence***(This is where the playlist located in the program mentioned in Chapter 2)***

**Snapshots of the codes for this chapter:**

First and foremost, we set our variable for the playlist.

Then we create a new file object in this named “folder” that represents the directory of the specified path. Then we retrieve the file objects that are contained in the folder directory. Then we make a DefaultListModel object called listModel, which is used to store a list of strings.

Then we make for loop to iterate over each file object.As for the if code I put it there so that it would only get the files that ends with a .wav extension since our music player only plays WAV files.

We then add the element file get name and make our JList with the same name as the variable.

***Code:***

A screen shot of a computer screen

Description automatically generated with low confidence

**References:**

*Cohen, S., & Shapira, O. (2017).* ***A Brief History of Music Consumption: A Field Experiment in Israel.*** *Journal of New Music Research, 46(1), 37-49.*

*Dredge, S. (2019, March 14).* ***How streaming saved the music industry. The Guardian.*** *Retrieved from* [*https://www.theguardian.com/technology/2019/mar/14/how-streaming-saved-the-music-industry*](https://www.theguardian.com/technology/2019/mar/14/how-streaming-saved-the-music-industry)

*Bell, G. (2022, February 22).* ***What Is a Music Player? Lifewire.*** *Retrieved from* [*https://www.lifewire.com/what-is-a-music-player-4686456*](https://www.lifewire.com/what-is-a-music-player-4686456)

***Techopedia. (n.d.). Music Player.*** *Retrieved from* [*https://www.techopedia.com/definition/22346/music-player*](https://www.techopedia.com/definition/22346/music-player)