```
package com.internshala.echo.fragments
import android.app.Activity
import android.content.Context
import android.media.AudioManager
import android.media.MediaPlayer
import android.net.Uri
import android.os.Bundle
import android.os.Handler
import android.support.v4.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageButton
import android.widget.SeekBar
import android.widget.TextView
import com.cleveroad.audiovisualization.AudioVisualization
import com.cleveroad.audiovisualization.DbmHandler
import com.cleveroad.audiovisualization.GLAudioVisualizationView
import com.internshala.echo.CurrentSongHelper
import com.internshala.echo.R
import com.internshala.echo.Songs
import java.util.*
import java.util.concurrent.TimeUnit
/**
 * A simple [Fragment] subclass.
 */
class SongPlayingFragment : Fragment() {
    var myActivity: Activity? = null
    var mediaPlayer: MediaPlayer? = null
    var startTimeText: TextView? = null
    var endTimeText: TextView? = null
    var playPauseImageButton: ImageButton? = null
    var previousImageButton: ImageButton? = null
    var nextImageButton: ImageButton? = null
    var loopImageButton: ImageButton? = null
    var shuffleImageButton: ImageButton? = null
    var seekBar: SeekBar? = null
    var songArtistView: TextView? = null
    var songTitleView: TextView? = null
    var currentPosition: Int = 0
    var fetchSongs: ArrayList<Songs>? = null
    var currentSongHelper: CurrentSongHelper? = null

    /*Declaring the variables for using the visualizer*/
    /*Audio Visualization used for the visual aspects of sound*/
    var audioVisualization: AudioVisualization? = null

    /*The visualization view*/
    var glView: GLAudioVisualizationView? = null

    var updateSongTime = object : Runnable {
        override fun run() {
            val getCurrent = mediaPlayer?.currentPosition
            startTimeText?.setText(String.format("%d:%d",
                    TimeUnit.MILLISECONDS.toMinutes(getCurrent?.toLong() as Long),

TimeUnit.MILLISECONDS.toSeconds(TimeUnit.MILLISECONDS.toMinutes(getCurrent?.toLong() as
Long))))
            seekBar?.setProgress(getCurrent?.toInt() as Int)
            Handler().postDelayed(this, 1000)
        }
    }
    override fun onCreateView(inflater: LayoutInflater?, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        val view = inflater!!.inflate(R.layout.fragment_song_playing, container, false)
        seekBar = view?.findViewById(R.id.seekBar)
```

```kotlin
        startTimeText = view?.findViewById(R.id.startTime)
        endTimeText = view?.findViewById(R.id.endTime)
        playPauseImageButton = view?.findViewById(R.id.playPauseButton)
        nextImageButton = view?.findViewById(R.id.nextButton)
        previousImageButton = view?.findViewById(R.id.previousButton)
        loopImageButton = view?.findViewById(R.id.loopButton)
        shuffleImageButton = view?.findViewById(R.id.shuffleButton)
        songArtistView = view?.findViewById(R.id.songArtist)

        /*Linking view with XML*/
        glView = view?.findViewById(R.id.visualizer_view)
        return view
    }
    override fun onViewCreated(view: View?, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        /*Connecting the audio visualization with the view*/
        audioVisualization = glView as AudioVisualization
    }
    override fun onAttach(context: Context?) {
        super.onAttach(context)
        myActivity = context as Activity
    }
    override fun onAttach(activity: Activity?) {
        super.onAttach(activity)
        myActivity = activity
    }
    override fun onResume() {
        super.onResume()

        /*When the fragment resumes, it resumes the visualization process*/
        audioVisualization?.onResume()
    }
    override fun onPause() {

        /*Pausing the visualization when fragment pauses to prevent battery drain*/
        audioVisualization?.onPause()
        super.onPause()
    }
    override fun onDestroyView() {

        /*Releasing all the resources held by the visualizer when fragment is removed*/
        audioVisualization?.release()
        super.onDestroyView()
    }
    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)
        currentSongHelper = CurrentSongHelper()
        currentSongHelper?.isPlaying = true
        currentSongHelper?.isLoop = false
        currentSongHelper?.isShuffle = false
        var path: String? = null
        var _songTitle: String? = null
        var _songArtist: String? = null
        var songId: Long = 0
        try {
            path = arguments.getString("path")
            _songTitle = arguments.getString("songTitle")
            _songArtist = arguments.getString("songArtist")
            songId = arguments.getInt("songId").toLong()
            currentPosition = arguments.getInt("position")
            fetchSongs = arguments.getParcelableArrayList("songData")
            currentSongHelper?.songPath = path
            currentSongHelper?.songTitle = _songTitle
            currentSongHelper?.songArtist = _songArtist
            currentSongHelper?.songId = songId
            currentSongHelper?.currentPosition = currentPosition
            updateTextViews(currentSongHelper?.songTitle as String,
currentSongHelper?.songArtist as String)
        } catch (e: Exception) {
            e.printStackTrace()
        }
        mediaPlayer = MediaPlayer()
        mediaPlayer?.setAudioStreamType(AudioManager.STREAM_MUSIC)
        try {
            mediaPlayer?.setDataSource(myActivity, Uri.parse(path))
            mediaPlayer?.prepare()
        } catch (e: Exception) {
            e.printStackTrace()
        }
        mediaPlayer?.start()
        processInformation(mediaPlayer as MediaPlayer)
        if (currentSongHelper?.isPlaying as Boolean) {
            playPauseImageButton?.setBackgroundResource(R.drawable.pause_icon)
        } else {
            playPauseImageButton?.setBackgroundResource(R.drawable.play_icon)
        }
        mediaPlayer?.setOnCompletionListener {
            onSongComplete()
        }
        clickHandler()

        /*Initializing the handler to handle the visual effects*/
        var visualizationHandler = DbmHandler.Factory.newVisualizerHandler(myActivity as
Context, 0)

        /*Linking the audio visualization with the handler*/
        audioVisualization?.linkTo(visualizationHandler)
    }

    fun clickHandler() {
        shuffleImageButton?.setOnClickListener({
            if (currentSongHelper?.isShuffle as Boolean) {
                shuffleImageButton?.setBackgroundResource(R.drawable.shuffle_white_icon)
                currentSongHelper?.isShuffle = false
            } else {
                currentSongHelper?.isShuffle = true
                currentSongHelper?.isLoop = false
                shuffleImageButton?.setBackgroundResource(R.drawable.shuffle_icon)
                loopImageButton?.setBackgroundResource(R.drawable.loop_white_icon)
            }
        })
        nextImageButton?.setOnClickListener({
            currentSongHelper?.isPlaying = true
            if (currentSongHelper?.isShuffle as Boolean) {
                playNext("PlayNextLikeNormalShuffle")
            } else {
                playNext("PlayNextNormal")
            }
        })
        previousImageButton?.setOnClickListener({
            currentSongHelper?.isPlaying = true
            if (currentSongHelper?.isLoop as Boolean) {
                loopImageButton?.setBackgroundResource(R.drawable.loop_white_icon)
            }
            playPrevious()
        })
        loopImageButton?.setOnClickListener({
            if (currentSongHelper?.isLoop as Boolean) {
                currentSongHelper?.isLoop = false
                loopImageButton?.setBackgroundResource(R.drawable.loop_white_icon)
            } else {
                currentSongHelper?.isLoop = true
                currentSongHelper?.isShuffle = false
                loopImageButton?.setBackgroundResource(R.drawable.loop_icon)
                shuffleImageButton?.setBackgroundResource(R.drawable.shuffle_white_icon)
            }
        })
        playPauseImageButton?.setOnClickListener({
            if (mediaPlayer?.isPlaying as Boolean) {
                mediaPlayer?.pause()
                currentSongHelper?.isPlaying = false
                playPauseImageButton?.setBackgroundResource(R.drawable.play_icon)
            } else {
                mediaPlayer?.start()
                currentSongHelper?.isPlaying = true
                playPauseImageButton?.setBackgroundResource(R.drawable.pause_icon)
            }
        })
    }
    fun playNext(check: String) {
        if (check.equals("PlayNextNormal", true)) {
            currentPosition = currentPosition + 1
        } else if (check.equals("PlayNextLikeNormalShuffle", true)) {
            var randomObject = Random()
            var randomPosition = randomObject.nextInt(fetchSongs?.size?.plus(1) as Int)
            currentPosition = randomPosition
        }
        if (currentPosition == fetchSongs?.size) {
            currentPosition = 0
        }
        currentSongHelper?.isLoop = false
        var nextSong = fetchSongs?.get(currentPosition)
        currentSongHelper?.songPath = nextSong?.songData
        currentSongHelper?.songTitle = nextSong?.songTitle
        currentSongHelper?.songArtist = nextSong?.artist
        currentSongHelper?.songId = nextSong?.songID as Long
        updateTextViews(currentSongHelper?.songTitle as String,
currentSongHelper?.songArtist as String)
        mediaPlayer?.reset()
        try {
            mediaPlayer?.prepare()
            mediaPlayer?.start()
            processInformation(mediaPlayer as MediaPlayer)
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
    fun playPrevious() {
        currentPosition = currentPosition - 1
        if (currentPosition == -1) {
            currentPosition = 0
        }
        if (currentSongHelper?.isPlaying as Boolean) {
            playPauseImageButton?.setBackgroundResource(R.drawable.pause_icon)
        } else {
            playPauseImageButton?.setBackgroundResource(R.drawable.play_icon)
        }
        currentSongHelper?.isLoop = false
        var nextSong = fetchSongs?.get(currentPosition)
        currentSongHelper?.songPath = nextSong?.songData
        currentSongHelper?.songTitle = nextSong?.songTitle
        currentSongHelper?.songArtist = nextSong?.artist
        currentSongHelper?.songId = nextSong?.songID as Long
        updateTextViews(currentSongHelper?.songTitle as String,
currentSongHelper?.songArtist as String)
        mediaPlayer?.reset()
        try {
            mediaPlayer?.setDataSource(myActivity, Uri.parse(currentSongHelper?.songPath))
            mediaPlayer?.prepare()
            mediaPlayer?.start()
            processInformation(mediaPlayer as MediaPlayer)
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
    fun onSongComplete() {
        if (currentSongHelper?.isShuffle as Boolean) {
            playNext("PlayNextLikeNormalShuffle")
            currentSongHelper?.isPlaying = true
        } else {
            if (currentSongHelper?.isLoop as Boolean) {
                currentSongHelper?.isPlaying = true
                var nextSong = fetchSongs?.get(currentPosition)
                currentSongHelper?.currentPosition = currentPosition
                currentSongHelper?.songPath = nextSong?.songData
                currentSongHelper?.songTitle = nextSong?.songTitle
                currentSongHelper?.songArtist = nextSong?.artist
                currentSongHelper?.songId = nextSong?.songID as Long
                updateTextViews(currentSongHelper?.songTitle as String,
currentSongHelper?.songArtist as String)
                mediaPlayer?.reset()
                try {
                    mediaPlayer?.setDataSource(myActivity,
Uri.parse(currentSongHelper?.songPath))
                    mediaPlayer?.prepare()
                    mediaPlayer?.start()
                } catch (e: Exception) {
                    e.printStackTrace()
                }
            } else {
                playNext("PlayNextNormal")
                currentSongHelper?.isPlaying = true
            }
        }
    }
    fun updateTextViews(songTitle: String, songArtist: String) {
        songTitleView?.setText(songTitle)
        songArtistView?.setText(songArtist)
    }
    fun processInformation(mediaPlayer: MediaPlayer) {
        val finalTime = mediaPlayer.duration
        val startTime = mediaPlayer.currentPosition
        seekBar?.max = finalTime
        startTimeText?.setText(String.format("%d: %d",
                TimeUnit.MILLISECONDS.toMinutes(startTime.toLong()),
                TimeUnit.MILLISECONDS.toSeconds(startTime.toLong()) -
TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes(startTime.toLong())))
        )
        endTimeText?.setText(String.format("%d: %d",
                TimeUnit.MILLISECONDS.toMinutes(finalTime.toLong()),
                TimeUnit.MILLISECONDS.toSeconds(finalTime.toLong()) -
TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes(finalTime.toLong())))
        )
        seekBar?.setProgress(startTime)
        Handler().postDelayed(updateSongTime, 1000)
    }
}
```